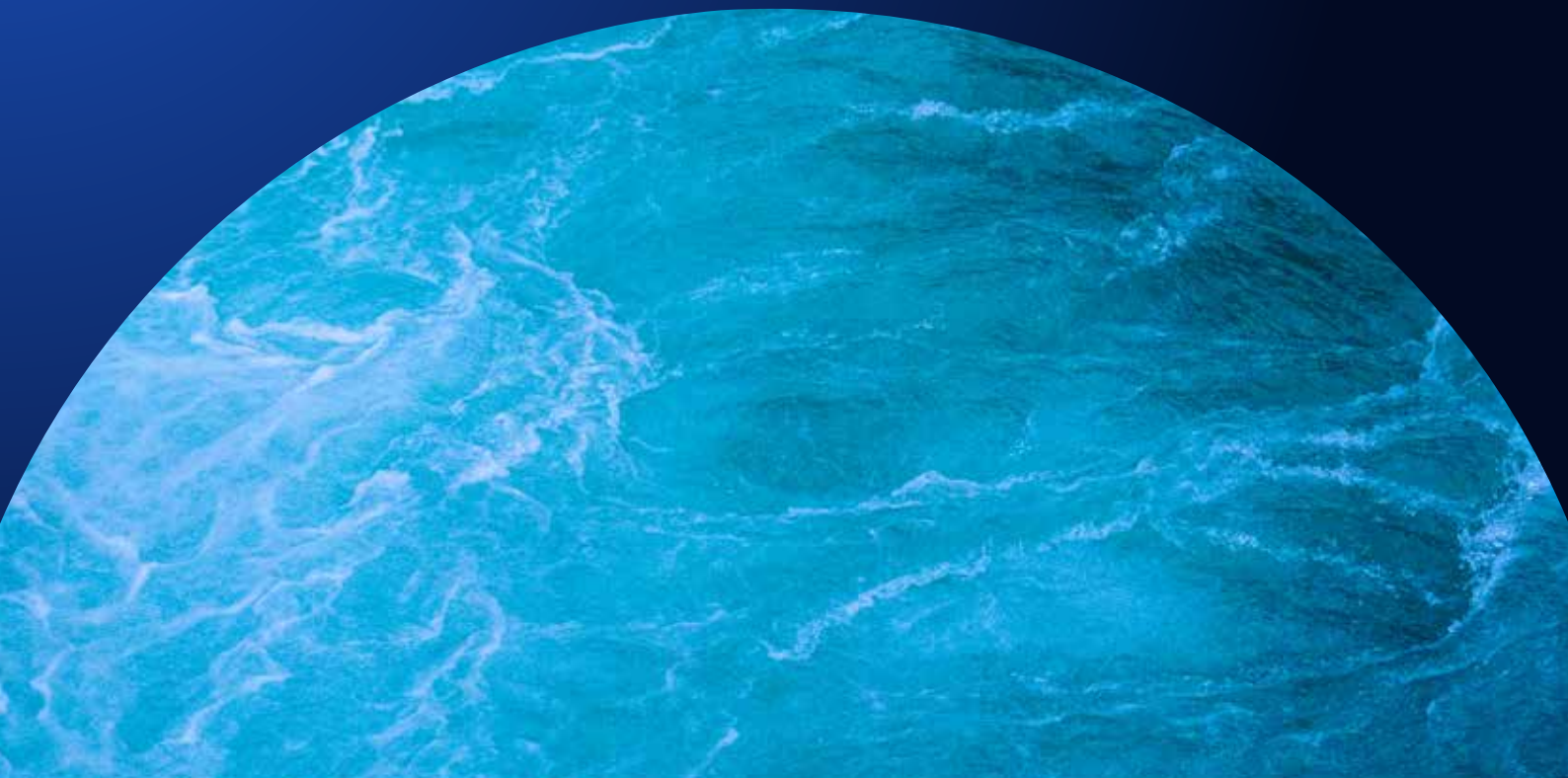
MARINE SENSOR INTEROPERABILITY APPLIED TO IN SITU OCEAN SOUND PROCESSING

PhD Thesis

ENOC MARTÍNEZ PADRÓ



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament d'Enginyeria Electrònica

Marine Sensor Interoperability Applied to in situ Ocean Sound Processing

Article-based PhD Thesis

Thesis submitted in partial fulfillment of the requirement for the PhD Degree issued by the Universitat Politècnica de Catalunya, in its Electronic Engineering Program

Enoc Martínez Padró

Supervisors:
Dr. Spartacus Gomáriz Castro
Dr. Joaquín del Río Fernández

June 2021


Enoc Martínez Padró, 2021

Marine Sensor Interoperability Applied to in situ Ocean Sound Processing

Enoc Martínez Padró
SARTI-MAR Research Group
Electronic Engineering Department
Universitat Politècnica de Catalunya
Rambla de l'Exposició 24, Edifici C, 08800
Vilanova i La Geltrú, Spain
<enoc.martinez@upc.edu>

Abstract

There has always been background sound in the oceans due to natural factors. However, since the first hydrophone deployments at mid-20th century, measured ocean sound levels have been steadily rising due to an increment of human activities such as shipping, construction, sonar and seismic exploration. Ocean sound is an important environmental factor for many species, especially to those using underwater sounds for localization and communication (in example marine mammals). The introduction of energy into the marine habitat in the form of underwater noise is harmful for the ecosystem. Therefore, it needs to be properly monitored to achieve a good environmental status.

The intrinsic nature of the marine environment complicates the acquisition of representative underwater sound datasets. Wide ocean extensions at different depths need to be monitored, presenting severe limitations on the available sampling methods. The deployment of fixed stations such as anchored autonomous recorders or moored hydrophone arrays are expensive and present numerous issues such as storage capacity, limited autonomy and low-bandwidth communications. On the other hand, state-of-the-art unmanned vehicles, such as underwater gliders or autonomous surface vehicles have an exceptional endurance, as well as excellent sampling capability over vast areas. However, these observation platforms are even more constrained in terms of power availability and communication's bandwidth.

Commercial off-the-shelf hydrophones used in ocean sound monitoring provide raw acoustic data, whether as acoustic recordings or as data streams. Due to the high volume of raw data produced by hydrophones, streaming this data to a shore station using constrained communications is generally not possible. Moreover, the telemetry is usually one of the most power-demanding systems in observation platforms. Thus, in order to obtain ocean sound levels, acoustic data has to be stored on-board and processed offline after the platform's recovery. However, if raw acoustic data could be processed in situ, the volume of data to be sent would be reduced by several orders of magnitude. Therefore, processing acoustic data in situ would allow real-time ocean sound monitoring from state-of-the-art observation platforms with constrained

resources.

Hydrophones, as well as the vast majority of marine sensors, do not have standardized interfaces, proving difficult to integrate into observation platforms. To overcome this lack interoperability, users develop their specific drivers, which is a time-consuming and error-prone task that requires in-depth knowledge of both the sensor and the observation platform. Furthermore, since environmental data should be shared across the scientific community to be properly analyzed, data has to be post-processed and formatted in order to be published in spatial data infrastructures. If a standardized approach is not considered, these operations require custom-made software components which increase the operation and maintenance costs of both acquisition systems and data infrastructures.

This thesis applies interoperability techniques to reduce the operational costs of ocean sound monitoring. First, a standards-based interoperable architecture to integrate hydrophones (and other marine sensors) into observation platforms is proposed. Standardized data and metadata management is carefully considered within this architecture, ensuring its re-usability and easing its coupling into spatial data infrastructures. An efficient, cross-platform algorithm for in situ ocean sound processing is provided. This algorithm allows state-of-the-art observation platforms to monitor ocean sound in real-time, regardless of the platform's constraints.

Keywords: Ocean Sound, Interoperability, Sensor Web Enablement, Passive Acoustic Monitoring, SensorML, Embedded Processing, Real-time Systems, Hydrophones.

Resum

Sempre hi ha hagut so de fons en els oceans degut a factors naturals. Tanmateix, des dels primers desplegaments d'hidròfons a mitjans del segle XX, els nivells de so mesurats als oceans han anat augmentant contínuament degut a l'increment d'activitats humanes com la navegació, la construcció, els sonars i les campanyes sísmiques d'exploració. El so en els oceans és un important factor ambiental per moltes espècies, especialment per aquelles utilitzant-lo per localització i comunicacions (per exemple mamífers marins). Per tant, la introducció d'energia a l'habitat marí ha de ser degudament mesurada.

La intrínseca naturalesa de l'entorn marí complica l'adquisició de conjunts de dades representatius sobre el so als oceans. Grans extensions d'aigua a diferents profunditats han de ser mesurades, limitant considerablement els mètodes de mostreig disponibles. El desplegament d'estacions fixes, com gravadors autònoms o conjunts d'hidròfons fondejats són cars i presenten problemes com la limitada autonomia o el reduït ample de banda disponible. Els vehicles no tripulats d'última generació, com els planadors submarins o vehicles de superfície autònoms, tenen una autonomia excepcional, al mateix temps que poden cobrir extenses àrees. Tanmateix, aquestes plataformes d'observació encara tenen més limitacions en termes d'energia i comunicacions.

Els hidròfons comercials utilitzats en el control del so als oceans, proporcionen dades acústiques en brut en gravacions o bé en transmissió contínua. Per tant, aquestes dades en brut han de ser processades a les estacions de terra. Degut al gran volum de dades produït pels hidròfons, la transmissió de les dades en brut utilitzant comunicacions amb ample de banda reduït no és possible. Malgrat això, si les dades es poguessin processar a bord de les plataformes d'observació, la quantitat d'informació a ser transmesa es reduiria en varis ordres de magnitud.

Els hidròfons, així com la gran majoria de sensors marins, no tenen interfícies de comunicacions estàndards, dificultant la seva integració en plataformes d'observació. Per a superar aquesta falta d'interoperabilitat, els usuaris desenvolupen programes específics, resultant en una tasca propensa als errors, lenta i laboriosa. Desenvolupar aquests programes requereix coneixement en profunditat tant de la plataforma

d'observació com del sensor. A més a més, les dades ambientals han de ser compartides entre la comunitat científica per a ser degudament analitzades. Per tant, aquestes dades han de ser post-processades i se'ls hi ha de donar un format adequat per a ser publicades en infraestructures de dades científiques. Si no s'utilitza una estratègia estandarditzada, aquestes operacions requereixen components fets a mida, augmentant significativament els costos d'operació i manteniment dels sistemes d'adquisició i les infraestructures de dades.

Aquesta tesis aplica tècniques d'interoperabilitat per a reduir els costos operacionals dels sistemes d'observació de so oceànic. Primer, es proposa una arquitectura universal, estandarditzada i interoperable per la integració d'hidròfons (i altres sensors marins) en plataformes d'observació marines. Dins d'aquesta arquitectura es posa èmfasis en la gestió de dades i metadades de manera estàndard, assegurant la reutilització, modularitat i capacitat d'acoblament a infraestructures de dades. Respecte al so submarí, es proposa un algoritme multiplataforma per al processat a bord del so oceànic. Aquest algoritme permet a les plataformes d'observació d'última generació realitzar mesures en temps real de so oceànic, superant les seves limitacions intrínseques.

Acknowledgements

I would like to thank all those people who in many different ways have helped me in the accomplishment of this PhD thesis.

First, I would like to thank my thesis supervisors Joaquín del Río and Spartacus Gomáriz for their continuous support. Although the situation hasn't been always easy, they always found a way to keep me within the team. Thus, thanks to their efforts, my dream of becoming a researcher has become a reality.

I would like to thank the whole SARTI team for their support, especially Daniel M. Toma and Carla Artero, who helped me a lot during the critical first stages of my thesis. I would also like to thank Albert García Benadí, who introduced me to underwater acoustics and helped me with all those complex equations and mathematical notations. Finally, I would like to thank Marc Nogueras, Matías Carandell, Ivan Masmitjà, David Sarrià and all other SARTI divers for making sure I didn't drown during my first clumsy dives.

Thanks to Eric Delory, Eduardo Caudet and the rest of PLOCAN team for their hospitality during my many visits at Gran Canaria.

I would also like to thank Bertrand Moreau, Julien Legrand, Nadine Lanteri, Jérôme Blandin and the rest of the Ifremer team for giving me a warm welcome during my internship at Brest .

I would like to thank Herve Precheur, Cassandra Díaz, Irene González and the rest of SensorLab team for believing in my work. Thanks to their incredible work with PIROS, the outcomes of this thesis have been materialized in a commercial system.

Also, I would like to thank Ehsan Abdi from Cyprus-Subsea Services Consulting for his enthusiasm and faith in my developments.

Finally, I would like to thank my family and friends, who patiently listened to my tech talk without understanding a single word. Without their support I never would have gotten this far.

Contents

Chapter 1

Introduction

1.1 Ocean Sound

There has always been background sound in the oceans due to natural factors. However, since the first hydrophone deployments at mid-20th century, measured underwater sound levels have been steadily rising due to an increment of human activities in the oceans [1]. Ocean sound is an important environmental factor, which affects numerous species. Thus, the introduction of energy in the form of underwater noise adversely affects the marine environment. Some consequences of underwater noise pollution in fish and marine mammals are: changes in the spatial distribution, adverse fitness consequences, communications masking and interferences in predator-prey interactions [2].

The term "underwater noise" is usually reserved for is for sound that is harmful to marine life. The term "ocean sound" refers to all underwater sound present in the underwater environment, regardless of its source or impact.

Ocean sound sources can be classified in two groups: anthropogenic sources (generated by human activities) and natural sources. Marine mammals are a prominent natural source, producing sounds in a wide range of frequencies, in example dolphin whistles and whale calls. Fish can also be a source of sound in different ways, such as stridulation and using muscles on the swim bladder. Fish do not only produce sounds as individuals, but also in groups, which are highly variable depending on the habitat, species and even the time of day. The increase in low-frequency sound can be as much as from 20 to 30 dB in the presence of chorusing fish [1]. Breaking waves and wind also affect the global level of the underwater sound by generating sea-surface agitation, especially in the mid-frequency band (500Hz-25 kHz). Wind related sound also affects the low frequency band (0 - 500Hz), but this band is usually masked by shipping noise [2].

Anthropogenic noise is generated by human activities such as commercial shipping, recreational boating, oil and gas exploration, naval operations (military sonars and communications), pile-driving and offshore energy parks. These sources are becoming both more pervasive and more powerful, increasing oceanic background sound levels as well as peak intensity levels [1].

The area over which anthropogenic noise may adversely impact marine species depends upon the propagation, duration and spectral characteristics of the sound. A sound that may affect the behaviour of some species may be outside the hearing range of others. Thus, in order to assess the impact of underwater noise in the marine habitat, ocean sound levels and spectral contents can be compared with hearing thresholds of different species. Numerous studies about the impact of different human activities can be found in the literature, such as construction and pile-driving [3], shipping [4–6] and offshore energy parks [7, 8].

The assessment and mitigation of underwater noise remains an open challenge, not only to protect the marine environment and ecosystem, but also to protect the resources of marine-related economic and social activities.

1.2 Marine Strategy Framework Directive

In order to achieve a "good environmental status" in European Waters, the European Commission approved the Marine Strategy Framework Directive (MSFD) [9]. As stated in the directive:
Good environmental status means the environmental status of marine waters where these provide ecologically diverse and dynamic oceans and seas which are clean, healthy and productive within their intrinsic conditions, and the use of the marine environment is at a level that is sustainable, thus, safeguarding the potential for uses and activities by current and future generations.

To achieve this goal, the directive defines a set of descriptors meant to monitor the environmental status, including monitoring of low-frequency continuous sounds (indicator 11.2). According to this indicator, continuous long-term ocean sound within the 63 and 125 Hz third-octave bands (center frequency) should be monitored across european regional waters in order to obtain statistically representative datasets [10]. Although the directive only specifies the 63 and 125 Hz third-octave bands, some experts suggested that these indicators may not be sufficient to achieve an accurate assessment of the soundscape and proposed to extend the monitored bands up to 20 kHz [11].

Despite advances in recent years, ocean sound is still undersampled and there is insufficient information on these indicators. There has been significant advances in

modelling and predicting underwater noise, but in situ measurements are required in order to calibrate and validate the models.

1.3 Ocean Sound Monitoring

As concern about underwater noise increased, numerous studies analyzing ocean sound have been presented [12–17]. Assessing the underwater sound levels in the ocean from raw hydrophone data is not straightforward. Some studies use the total power of the signal [14]. On the contrary, others use a more complex approach, like averaged spectral density [6, 18]. Special care must be put on the selected techniques, since the achieved underwater sound levels may vary significantly [19].

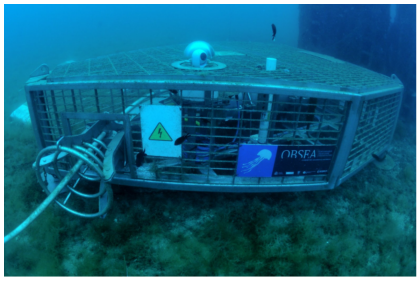
Commercial off-the-shelf hydrophones used in ocean sound monitoring usually provide raw acoustic data, whether as acoustic recordings or as a data stream. Thus, acoustic data has to be post-processed to obtain meaningful underwater sound level values. Due to the nature of the marine environment, marine sensors (including hydrophones) are usually deployed in observation platforms.

1.3.1 Marine Observation Platforms

There are a wide range of state-of-the-art observation platforms able to host marine sensors, such as cabled observatories [20], Autonomous Surface Vehicles (ASV) [21, 22], moored buoys and underwater gliders [23] (figure 1.1). Among other tasks, these platforms supply power to sensors, configure them, gather sensor data and transmit this data to shore stations. From the sensor integration point of view, the most important parameters that need to be considered are telemetry and power availability.

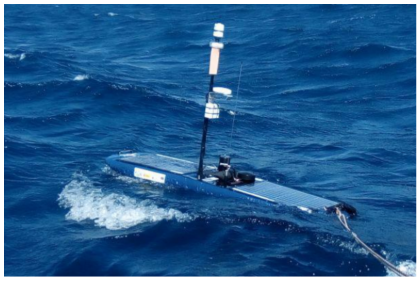
The telemetry is the technology used by an observation platform to send acquired data to communicate with a shore station. Some common telemetry technologies are broadband Ethernet for cabled platforms, Global System for Mobile communications (GSM) in coastal areas and costly satellite communications for open-ocean applications. Some platforms, such as underwater gliders, only have telemetry during short periods of time (surface time), further restricting the telemetry. The telemetry technology used has important implications on the amount of data that can be sent (bandwidth), the economic cost of transmitting data and also the energy used for this transmission.

The power availability determines the autonomy of an observation platform and the amount of data that can be transmitted. Furthermore, the power availability can also limit the computational capacity of the observation platform. In example, underwater gliders generally use ultra-low power microcontrollers to save energy and extend their autonomy.



(a) **Cabled Observatories**
Power availability: Very High
Telemetry: Broadband Ethernet

(b) **Moored Buoys**
Power availability: Medium
Telemetry: GSM / satellite



(c) **Autonomous Surface Crafts**
Power availability: low
Telemetry: satellite

(d) **Underwater Gliders**
Power availability: ultra-low
Telemetry: satellite (intermittent)

Figure 1.1: Telemetry and power availability of four observation platforms: (a) cabled observatory, (b) moored buoy, (c) Autonomous Surface Vehicle and (d) underwater glider. (Source d: ALSEAMAR-ALCEN, source b: PLOCAN).

1.3.2 Ocean Sound Measurement Systems

Commercial off-the-shelf hydrophones used in ocean sound monitoring usually provide raw acoustic data, whether as acoustic recordings or as a data stream. Thus, acoustic data has to be post-processed to obtain meaningful underwater sound values. Due to the high sampling rate used by hydrophones (usually from tens to hundreds of kHz), streaming raw acoustic data from an observation platform to a shore station is only possible with broadband communications.

Cabled observatories equipped with hydrophones are an excellent observation platform for long-term ocean sound monitoring, since they usually do not have bandwidth nor power constrains [4,5,24]. However, these infrastructures are scarce and costly to maintain. Some surface buoys equipped broadband radio communications are also capable of streaming acoustic data in real-time to shore station for real-time processing [25]. However, the autonomy of these systems is greatly constrained by their power availability. Moreover, these buoys need to be located close to shore, since broadband radio link has limited coverage.

Moored autonomous recorders composed of one or several hydrophones connected

to a recording unit have also been used in large-scale deployments for ocean sound monitoring [13, 15, 26]. The autonomy of such devices is mainly reduced by two factors: power and storage capacity. The storage of acoustic data is not trivial, since it can be up to several gigabytes per day depending on the sampling rate. Usually data acquired by these devices is only available after recovery, so they do not provide real-time capabilities.

Traditional autonomous platforms such as Autonomous Underwater Vehicles (AUVs) rely on thrusters as propulsion mechanism. Although AUVs are widely used in oceanographic applications, their thrusters are noisy by nature. Thus, they are not suitable for ocean sound monitoring applications.

Underwater gliders are similar to AUVs, but instead of thrusters they use buoyancy control as propulsion mechanism. Although the pumps and motors that conform the buoyancy control produces some self-noise in short intervals, they are much quieter than other autonomous vehicles [27, 28]. Thus, they have been used to sense the underwater soundscape [29, 30].

Underwater gliders have intermittent communication link to shore stations during surface time. However, their satellite communications have a very limited bandwidth, not suitable for raw acoustic data transmission.

Telemetry is usually one of the more power-demanding components of an autonomous system such as underwater gliders. Thus, reducing the data transmission (and its associated power consumption) is vital to extend their autonomy.

If the acoustic data could be processed in situ, the amount of information to be transmitted will decrease by several orders of magnitude, allowing real-time measurements from autonomous platforms with telemetry, such as gliders, profilers, moored buoys, etc. However, processing acoustic data in real-time is not a trivial task due to the required computational power and intrinsic complexity of acoustic signals. Although there are some hydrophone prototypes with embedded underwater sound level algorithms, they are not yet widely used, and their embedded algorithms still have room for improvement [31].

1.4 Interoperability and Standardization

Processing ocean sound data on-board observation platforms is not the only challenge in ocean sound monitoring. Integrating hydrophones into observation platforms is also costly and time-consuming task. Hydrophones, as the vast majority of marine instrumentation, tend to use proprietary and non-standardized communication protocols for their instruments. Moreover, even in the same instrument family, different versions of the instrument's firmware may present significant differences in

the protocol. Therefore, if a new instrument has to be integrated or it has to be moved from one platform to another, new specific software components have to be developed. Although manufacturers usually provide drivers for desktop environments, marine sensors are frequently integrated in embedded systems where the provided drivers cannot be used [32]. Generating a specific driver for each instrument-platform combination is a time-consuming task that requires in-depth knowledge of both sensor's protocol and the observation platform's architecture [32].

When global events are studied, data from different sources is often needed. This data is usually acquired by different sensors, deployed across wide areas, conforming complex sensor networks [33]. Due to the large number of sensor manufacturers and communication protocols, integrating heterogeneous sensors into data infrastructures is a complex task. Data and metadata management vary significantly across different scientific domains. Furthermore, different institutions focusing in the same domain may also use different and non-compatible formats, standards and interfaces. This heterogeneity leads to information silos, preventing the data to be effectively shared across different scientific communities.

As there are more and more data sources, the number of custom software components required to manage them increases, as well as scientific data infrastructures maintenance costs. In example, when studying ocean sound over a specific area, acoustic data may be acquired using different observation platforms: underwater cabled observatories, autonomous recorders, underwater gliders, etc. Different hydrophones (with their own proprietary protocols) need to be integrated into platforms, each one with different hardware/software architectures. Then, acoustic data has to be processed, some may be processed in real-time while others need to be processed off-line after the platform's recovery. Finally, once all acoustic data is acquired, it has to be merged and compared with possible sound sources, such as shipping, wind speed data, etc. If common standards are not used, all this data has to be collected, transmitted, processed and harmonized manually.

Therefore, improving interoperability among the components within scientific data infrastructure should be a matter of great concern. Interoperability can be defined as the ability of two systems to exchange information and to interpret the information that has been exchanged [34]. A coherent infrastructure is needed to treat sensors in an interoperable, platform-independent and uniform way [35].

Interoperability can be separated in two different layers: syntactic and semantic. The syntactic interoperability is the ability of two systems to understand their formats and interfaces, allowing the flow of information, while semantic interoperability focuses on providing unambiguous meaning to the information that has been exchanged.

Different architectures have been presented to integrate heterogeneous sensor

resources into spatial data infrastructures, providing a resource independent view [33, 36]. However, due to the lack of standardization they are platform-specific and present problems when trying to integrate data and resources into other architectures. To avoid these issues a coherent, standardized and well-defined structure is needed.

1.4.1 Sensor Web Enablement Framework

The lack of standardization and data harmonization across scientific domains and data infrastructures has been the driving force for the Open Geospatial Consortium (OGC) to propose the Sensor Web Enablement framework (SWE) [37]. This framework is conformed by set of standards that define data models, encodings and common interfaces which aim to provide the building blocks for interoperable Sensor Web infrastructures. In this context, the concept of Sensor Web refers to a set of Web accessible sensor networks and their data/metadata that can be discovered and accessed using standard protocols and application programming interfaces [38]. The overall goal of SWE technologies and services is to ensure that geospatial data follows the FAIR principles: Findable, Accessible, Interoperable and Reusable [39, 40].

The SWE framework aims at tackling a huge interoperability challenge for middleware approaches since heterogeneous devices are expected to collaborate together in communication and information exchange. An overview on the SWE stack of components and services for marine data is depicted in 1.2. The SWE framework provides a good solution for data interoperability at the service level. However, implementations of SWE standards across different domains have found difficulties when integrating sensor resources into SWE services [41].

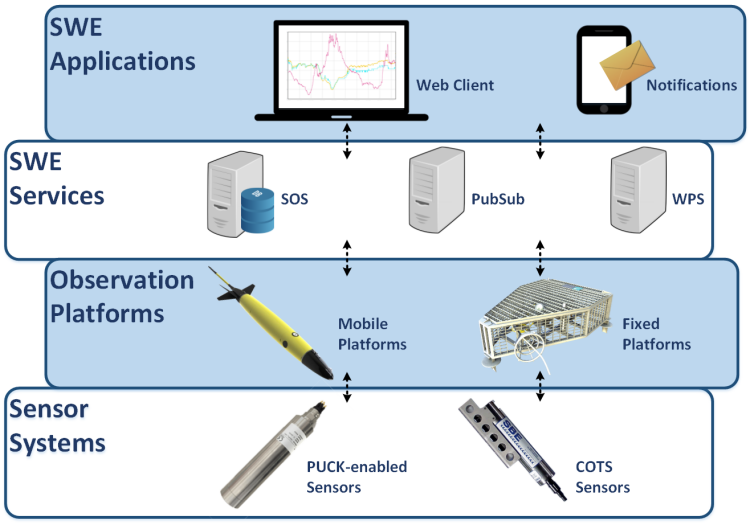


Figure 1.2: Sensor Web Enablement Stack of components and services for marine observation systems

Due to the nature of the marine environment, marine sensors (including hydrophones) are usually deployed in observation platforms. Thus, in the marine domain, sensor integration into SWE infrastructures is even more challenging, since sensors are not directly coupled with SWE services. Instead, they are integrated into observation platforms, adding an intermediate layer between the sensor and the services, as depicted in figure 1.2. This extra layer increases complexity when integrating marine sensors into data infrastructures due to the heterogeneity of marine observation platforms.

1.4.2 Sensor Integration Strategies

When integrating marine sensors into data infrastructures, two different steps have to be taken into account. The first is the operational interoperability, ensuring that the observation platform can interface and acquire sensor data. Once the data has been acquired, the second step is to inject data (alongside with its associated metadata) into data infrastructures. The use of common data interfaces and formats is required to allow the flow of information. Additionally, semantic interoperability should also be addressed, ensuring that the transmitted data is correctly understood, processed and classified by data infrastructures. A good semantic framework provides the ability to share scientific data unambiguously across different infrastructures, allowing data management, curation and processing in an automated manner.

There have been some attempts to propose standardized communications interfaces for scientific instruments, such as the IEEE 1451 standard [42]. Although there was some initial optimism in this standard, manufacturers did not widely implement it and ultimately fell into disuse. However, the short-lived IEEE 1451 standard provided a valuable lesson: manufacturers are not willing to modify their communications protocols. On the contrary they tend to stick to their own proprietary protocols.

In real world applications a huge variety of sensor protocols, whether standardized or proprietary, are used in commercial instruments. Thus, another approach to address integration challenges is to abstract communication protocols using a common abstraction interface. The Sensor Abstraction Layer (SAL) attempts to describe all the sensors in a uniform manner [43]. It uses and expands the SensorML 1.0 standard to describe sensor interfaces and commands. It hides the specific technological details by matching sensor-specific commands to generic SAL commands. However, it still relies on a plug-in approach to integrate new sensors, which is only solves partially the integration challenges [44].

Other approaches using standardized components have been proposed, such as the use of lightweight SWE connectors (custom drivers) for bridging from sensor-specific protocols to SWE services [41]. This approach has been widely used in different

scenarios, such as environmental monitoring, precision agriculture and defence [45, 46]. However, a specific component for each sensor has to be generated ad hoc.

Similar to SAL, the Sensor Interface Descriptor (SID) model extends SensorML 1.0 to formally describe a sensor's protocol [47, 48]. The generated sensor description is used by a platform-independent driver (SID Interpreter), which translates between sensor protocol and Sensor Web protocols, effectively injecting data SWE infrastructures. A benefit of SID is the availability of open-source components to generate SID descriptions [49] as well as a SID interpreter middleware based on Java. Several applications of SID can be found at the literature [32, 50, 51].

However, a remaining open challenge is to practically include such a universal approach in abstracting the heterogeneity of marine sensors and observation platforms. Although a Java-based middleware may be an appropriate approach to abstract software on the operating system level, it may be not applicable to resource-constrained platforms. Furthermore, the SID model extension defines the whole Open Systems Interconnection (OSI) model stack for each sensor command, resulting in very verbose XML files, which are rather difficult to interpret.

A slightly different approach is to enhance each sensor controller with embedded Web services in order to archive and exchange data in a standardized manner with the components in the upper layers of the Sensor Web Enablement stack. This sensor integration strategy has been used in different fields, such as precision agriculture applications [46, 52, 53] and air quality monitoring [54, 55]. Although this approach provides a good interoperability with other Sensor Web components, it does not directly solve the challenge of integrating new sensors into a platform as it still relies on custom drivers. Furthermore, it may not be applicable to certain marine observation platforms with limited computational and bandwidth resources.

Despite all the sensor integration strategies previously discussed, none of them is universally applicable to all marine instruments and marine observation platforms. Thus, the integration of marine sensors into observation platforms and the management of the acquired data into data infrastructures is still an open challenge, especially in hydrophones.

1.5 Thesis Objectives

Hydrophones, as well as the vast majority of marine sensors, do not have standardized communication protocols, proving difficult and time-consuming to integrate into observation platforms. To overcome this lack of interoperability, users and manufacturers need to develop their specific drivers, which is a time-consuming task that requires in-deep knowledge of both the instrument and the observation platform. In

the literature, there is a lack of standard tools to integrate marine sensors, including hydrophones.

State-of-the-art low-power observation platforms such as underwater gliders and unmanned surfaces vehicles are ideal tools for long-term scientific deployments at sea, providing a very good temporal and spatial coverage. However, due to their constrained nature, these platforms are not able to stream acoustic data in real-time to shore stations. Thus, ocean sound monitoring is usually performed offline, once data is recovered. If acoustic data could be processed in situ, the amount information to be transmitted would decrease by several orders of magnitude, allowing the monitoring of ocean sound in real-time. In the literature there is a lack of tools to perform in situ ocean sound processing from resource-constrained observation platforms.

This thesis aims to contribute to in situ ocean sound measurements using state-of-the-art marine observation platforms, with strong focus on using standard interoperable technologies. In order to achieve this goal, three objectives are defined (as depicted in figure 1.3):

- **Objective 1:** Improve existing standards-based metadata models to abstract marine sensor characteristics, protocols and interfaces, with emphasis on hydrophones.
- **Objective 2:** Contribute to plug-and-play strategies for sensor integration into state-of-the-art observation platforms, based on standardized sensor definitions.
- **Objective 3:** Optimization of algorithms for in situ ocean sound monitoring from resource-constrained platforms.

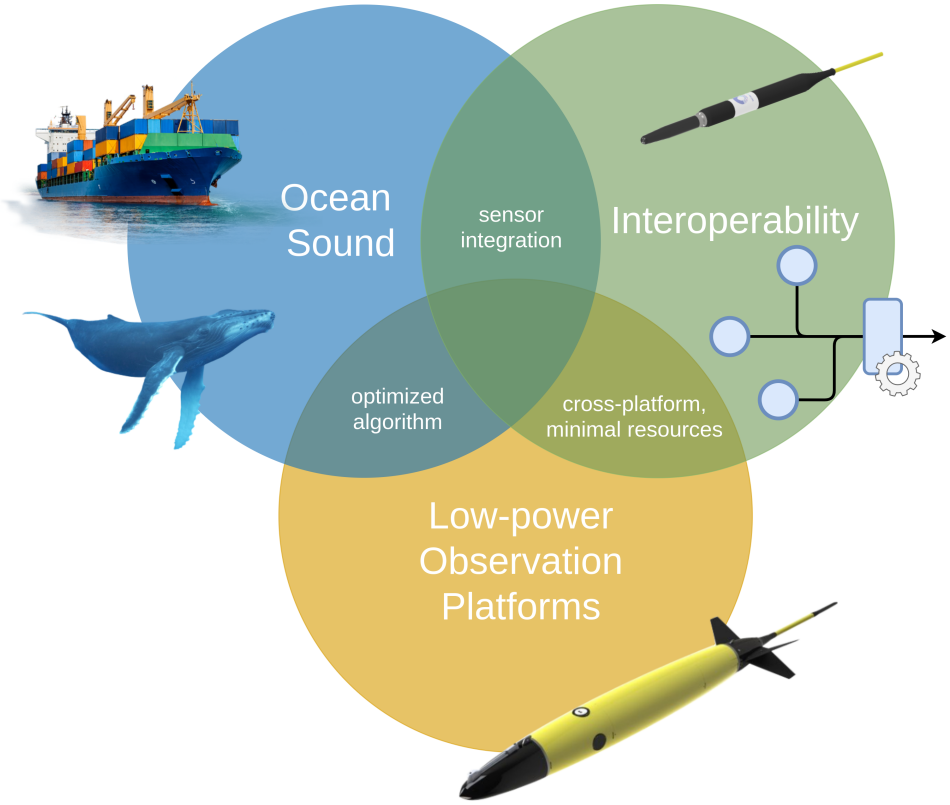


Figure 1.3: Venn diagram with the thesis objectives

1.6 Thesis Contributions

This thesis contributes at three different levels: sensor metadata modelling, a tool for plug-and-play sensor integration and an algorithm for in situ ocean sound monitoring.

The contribution to sensor metadata provides a model to unambiguously define sensor characteristics, interfaces, communications protocols and other sensor-related features. This model builds on existing standards and focus on machine-to-machine interactions.

The second contribution is a plug-and-play integration tool, able to interface and manage sensors regardless of its protocols or interface. This tool interprets sensor metadata descriptions to abstract sensor characteristics, acquire data and inject it into spatial data infrastructures.

Finally, an in situ ocean sound algorithm optimized for resource-constrained observation platforms is provided. This algorithm is embedded into the previously mentioned integration tool, achieving a universal end-to-end ocean sound measurement system, regardless of the underlying sensor or observation platform.

1.7 Dissertation Structure

The rest of this thesis is structured as follows:

- In chapter 2 a universal architecture for sensor integration is presented, addressing standardized sensor metadata descriptions (objective 1) and plug-and-play integration strategies (objective 2).
- In chapter 3 different algorithms for ocean sound monitoring are analyzed. Then, the most optimal is selected, implemented and verified fulfilling objective 3.
- In chapter 4 several deployments in real-world scenarios using the proposed architecture and ocean sound algorithm are presented and discussed.
- Finally, in chapter 5 some conclusions and future lines of research can be found.

Additionally, in appendix A there is a copy of the publications that conform this thesis. Article A.1 and conference papers A.3 and A.4 address the thesis' objectives 1 and 2 for generic sensors. Article A.2 addresses objectives 1, 2 and 3 with special emphasis on hydrophones.

Finally, in appendix B the user manual of the universal sensor integration tool (SWE Bridge) developed within this thesis can be found.

Chapter 2

Interoperable Architecture for Sensor Integration

2.1 Sensor Integration Requirements

As discussed in the state of the art, there is an urgent need in the marine observing community to provide solutions to integrate sensors (including hydrophones) into marine observation platforms. The physical layer is generally standardized and off-the-shelf sensors have well-known interfaces such as RS232, RS485 or Ethernet. However, scientific marine sensors tend to use proprietary protocols, making it difficult to adopt a common integration technique. Thus, in the marine context it is clear that sensors and platforms are not interoperable. This lack of interoperability is usually addressed using ad-hoc drivers for each sensor deployment. However, generating ad-hoc software components for each deployment is costly, error-prone and time-consuming.

Within this thesis, a generic architecture for plug-and-play sensor integration with minimal human intervention is proposed. In other words, an architecture to achieve a high degree of interoperability. Although the proposed architecture is generic and suitable for virtually any domain, it has a strong focus on marine instrumentation, specially hydrophones.

Interoperability can be split in two different layers: semantic and syntactic (as explained in paper A.2, section III). Syntactic interoperability is the ability of two systems to understand their formats and interfaces, allowing the flow of information. On the other hand, semantic interoperability focuses on providing unambiguous meaning of the information that has been exchanged.

The proposed architecture does only integrate a sensor into a hosting observation platform, but also provide seamless integration into SWE-based services. Thus, it provides an end-to-end data and metadata workflow, from the sensor to spatial data

infrastructures. In order to achieve integration in complex data infrastructures, it is not enough to establish a data stream. It is also required to provide unambiguous and machine-understandable definitions for each concept in order to achieve semantic interoperability.

As discussed in the paper A.1, the following requirements are identified in order to achieve a plug-and-play sensor integration mechanism:

1. **Sensor Detection**: Detect a new sensor when it is attached to an observation platform.
2. **Sensor Identification**: Obtain an unambiguous description of the sensor, including all the metadata to identify the sensor.
3. **Sensor Configuration**: This requirement addresses all operations required before the platform can start retrieving data from a sensor.
4. **Measurements Operations**: Retrieve sensor data, whether querying it directly or handling data streams.
5. **Sensor Registration**: Registering sensor metadata to a data infrastructure
6. **Data Ingestion**: Ingest sensor data to a data infrastructure
7. **Constrained Resources**: Ability to work in low-bandwidth, low-power and computationally-constrained scenarios.

2.2 Standards and Protocols

Instead of re-inventing the wheel by using custom-made mechanisms, the proposed architecture is entirely based on existing standards and protocols. In order to fulfil the previously presented requirements, a set of protocols and standards are presented in this section. The majority of these standards are part of the Open Geospatial Consortium's (OGC) Sensor Web Enablement framework (SWE). Figure 2.1 summarizes the standards used within this architecture and the requirements they aim to fulfil.

2.2.1 OGC PUCK Protocol

Sensor manufacturers tend to use proprietary and non-standardized communication protocols. Furthermore, even within different versions of the same sensor's firmware, the communication protocol has significant differences. Scientific and oceanographic sensors are generally manufactured by small and medium-sized companies, each

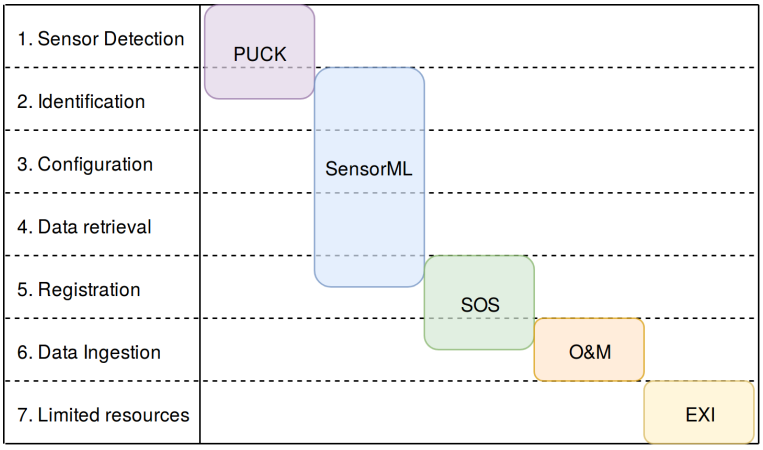


Figure 2.1: Requirements for a plug and play mechanism (rows) and protocols/standards that can fulfil these requirements (columns).

one with their proprietary communications. Thus, integrating marine sensors into observation platforms is a complex and time-consuming task due to the heterogeneity and lack of standardization.

The OGC PUCK protocol was born from the need of a common layer of communications across different scientific instruments [56]. On the contrary to other attempts to standardize sensor interfaces such as IEEE 1451 standard, it does not impose a full and complex protocol to sensor manufacturers. Thus, instead of replacing their proprietary protocols, it is a simple and small add-on to their existing protocols.

The OGC PUCK defines two modes of operation: *instrument mode* and *puck mode*. The instrument mode is the generic, proprietary operation of a given sensor. When a special command sequence is sent, called *softbreak*, the sensor changes to *PUCK mode* (state diagram is depicted in figure 2.2). In this mode, the sensor replies to the commands from the OGC PUCK protocol, which is a very simple set of 10 commands. These command set provide easy and transparent access to a persistent memory embedded within the sensor. The rationale behind this embedded persistent memory, is the need to store metadata in a standardized fashion within the sensor itself.

Within this persistent memory, named PUCK Payload, there are two different zones: the PUCK datasheet and the PUCK Payload. The PUCK datasheet is a simple 96-byte memory containing information such as UUID, manufacturer code, sensor model, sensor name, etc. The PUCK Payload is a memory where the operator can embed any information.

A few manufacturers such as RBR, Nortek, WETLabs and Sea-Bird Scientific integrated the OGC PUCK Protocol in some of their sensors. However, the majority manufacturers have been reluctant to include the OGC PUCK into their sensors. As

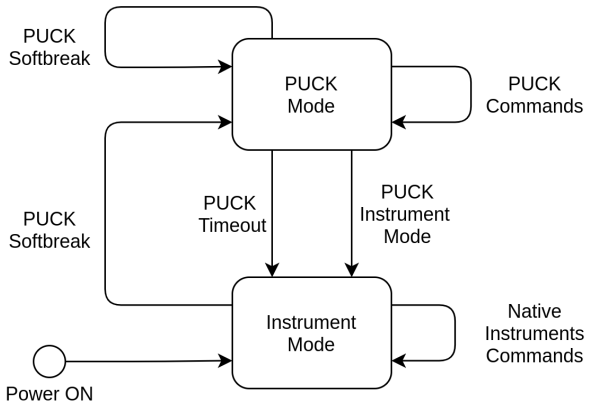


Figure 2.2: OGC PUCK protocol state diagram

a workaround, there have been several implementations of external PUCK devices, such as the Smart Cable (see section 4.5.2). This device acts as a man-in-the-middle, granting transparent communications until a *softbreak* is detected. Then, it switches to PUCK Mode, allowing access to the memory stored within the Smart Cable itself.

2.2.2 SWE Common Data Model

The OGC SWE Common Data Model provides a robust, unified and machine-understandable way to model and encode data and data streams into XML documents [57]. It defines several types of data, from simple scalar values up to complex arrays and data matrix. It allows encoding of ASCII data, binary data and even XML data streams. Within this work, this standard has been used to model and encode the arbitrary commands and data streams from sensors.

Although SWE Common Data Model is rarely used on its own, it is largely used within SensorML documents to encode sensor data streams.

2.2.3 Sensor Model Language

The OGC Sensor Model Language (SensorML) provides a framework to encode descriptions of sensors and sensor-related processes within XML files [58]. Its main goal is to enhance interoperability, making sensor descriptions understandable by machines and shareable between intelligent nodes. Moreover, additional information related to specific deployments can also be encoded using this standard. Thus, both sensor configuration, measurement operations and contextual information can be defined with the SensorML standard.

It is highly flexible and modular as it can describe almost every property related to a sensor or sensor-related process. However, this flexibility and modularity can prove a double-edged sword, as the same feature can be encoded in different ways,

increasing the difficulty to generate smart processes capable of interpreting SensorML definitions [59].

By combining the OGC PUCK protocol and a SensorML description file, it is possible to automatically detect a sensor and retrieve its description encoded in a single standardized file without any a-priori knowledge about the sensor. If an interpreter software understands this file, it can identify the sensor, configure it and retrieve its data, meeting the requirements 2, 3 and 4.

This standard provides a robust skeleton to encode sensor and sensor-related metadata in a machine-understandable way, providing the building blocks to achieve syntactic interoperability.

2.2.4 Observations & Measurements

The Observations and Measurements (O&M) standard specifies an abstract model as well as XML encoding for observations and related data, such as features involved in the sampling process [60]. It provides a uniform and unambiguous way to encode sensor measurements, fulfilling the requirement 6. Within this standard, and observation is defined an action whose result is an estimate of the value of some property of the feature of interest (entity being measured), obtained using a specified procedure.

O&M provides an abstract model for observations, which is very flexible, but at the same time can be confusing due to the abstraction level and terminology used. Figure 2.3 provides an example to illustrate how a simple temperature observation is modelled using this standard. Within this work all observations are in situ, thus, some conventions are used to reduce the abstraction level:

- The procedure is always the sensor that provided the observation.
- In fixed observation platforms, the feature of interest (FOI) is the station where the observation was taken, e.g. OBSEA Cabled Observatory.
- In mobile platforms (gliders, unmanned vehicles, etc.), the feature of interest (FOI) is the mission or campaign under which the observation was taken.
- The observed property refers to the variable that is being measured, e.g. sea water temperature.

Although O&M may be complex and confusing at first glance, it is a very powerful standard that provides well-defined and unambiguous information about all elements involved in the sampling process.

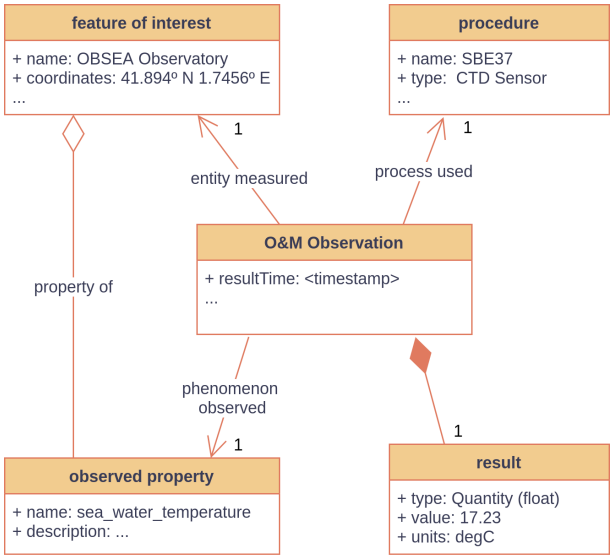


Figure 2.3: Example of a O&M Observation, where the sea water temperature (in degrees Celsius), is measured at the OBSEA Underwater Observatory using a SBE37 CTD.

2.2.5 Sensor Observation Service

The Sensor Observation Service (SOS) standard defines a Web service interface which provide means to register sensor data and metadata. Further, this standard also provides means to query data, metadata as well as representation of observed features [61].

It has a strong focus on machine-to-machine interactions, allowing the automated data and metadata management in an interoperable manner. Due to its role as data and metadata archive and access service, this standard is a key piece of Sensor Web infrastructures.

It uses the O&M standard to encode observations, thus it is highly versatile and provides robust relationships. Procedure descriptions (sensors), as well as features of interest (measured entities) are defined using the SensorML standard.

2.2.6 Efficient XML Interchange

The Efficient XML Interchange (EXI) is a World Wide Web Consortium's (W3C) standard candidate that enables the compression of large ASCII XML documents into efficient binary files, significantly reducing their size. This format has been designed to allow information sharing between devices with constrained resources, meeting requirement 7 [62].

EXI uses several strategies to reduce the redundancy in XML files, such as the string table to minimize redundancy. XML usually uses the same string within a start element tag (SE) and its associated end element tag (EE). In EXI, when a string is

found for the first time an identifier is assigned to it. Subsequent instances of the same string use only the identifier, not the string itself.

Another mechanism that it uses is the bit-packing technique. Instead of align the information at the byte level, it uses only the exact number of bits required for each element. For example, if a set of identifiers range from 0 to 14, instead of using 1 byte (0-255) it uses only 4 bits (0-15).

This format is an ideal for situations where each byte counts, such as constrained and costly satellite communications, where the company charges platform operators for every transmitted byte. However, the complexity of the generated files is increased and dedicated processes for encoding/decoding are required.

2.3 Architecture

Within this thesis a standards-based architecture for sensor integration is proposed. Its main goal is to provide a framework to integrate marine sensors (including hydrophones), achieving an end-to-end data and metadata workflow, fulfilling the thesis objective 2. Such architecture should be able to handle the heterogeneity of both marine sensors and marine observations platforms. Special focus is put on the use of existing standards, avoiding custom-made solutions.

Metadata plays an important role during the data life-cycle, providing vital context to observations: what was measured, where it was measured, who lead the deployment, etc. However, metadata is usually compiled after data is acquired and targets only the measurements themselves, providing little information of the overall acquisition chain. Within this work, metadata is not only used to provide contextual information, but also to configure generic data acquisition chains. In other words, the data acquisition, processing and storage processes are driven by the sensor's metadata. The metadata-driven architecture concept is described in paper A.2, section II, and applied throughout this thesis.

Both semantic and syntactic interoperability are carefully considered in this architecture. Firstly, syntactic interoperability is achieved by interfacing sensors on-the-fly based on their SensorML descriptions. Data and metadata in the architecture are enriched using state-of-the-art semantic web technologies, ensuring that further process and components are able to automatically understand the context, achieving semantic interoperability.

The proposed interoperable, metadata-driven architecture is depicted in figure 2.4. The first step in this architecture are scientific sensors. Each one has its own SensorML description, containing relevant metadata such as interface, communication protocol, configuration parameters, etc. Moreover, it may also contain acquisition-

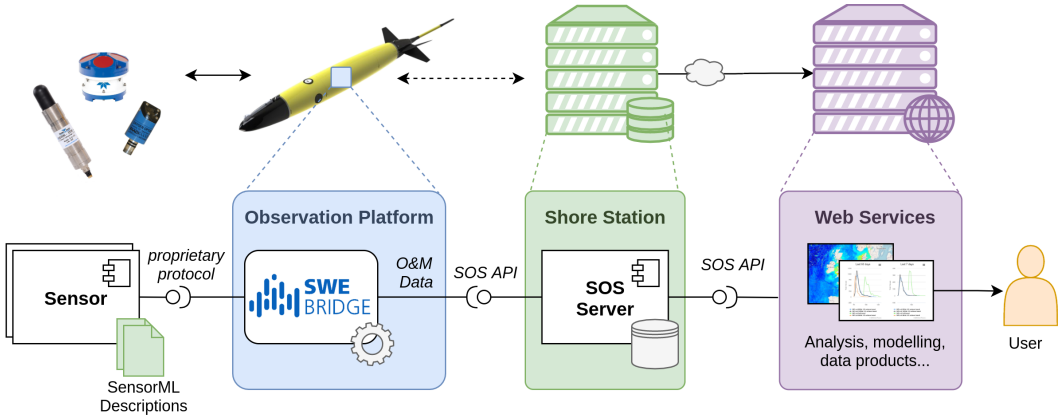


Figure 2.4: Interoperable Architecture for Sensor Integration diagram

specific information such as which format should be used to encode data, calibration procedures to be applied, etc. Sections 2.3.1 and 2.3.2 provide a detailed insight of SensorML description files. The concepts encoded within these SensorML descriptions are semantically annotated to ensure that both human and machines can understand their meaning and context.

The next step in the architecture is the SWE Bridge, an open-source, cross-platform universal driver developed within this thesis. This universal driver is able to decode spcecial SensorML files assigned to each sensor and configure an acquisition chain accordingly (see sections 2.3.1 and 2.3.2). Thus, it can integrate sensors into observation platforms on-the-fly. The mechanism to retrieve each SensorML file is discussed in detail in section 2.3.3. The SWE Bridge does not only provide integration and data acquisition mechanisms, it also has several modules that implement a wide range of functionalities, such as data formatting, real-time underwater sound processing, calibration, monitoring services, power management and more. A detailed overview of the SWE Bridge software can be found in section 2.4 and in its user manual, in appendix B.

The SWE Bridge aims to be deployed within observation platforms, so it is decoupled from the platform's communication link with the shore station. This is crucial in observation platforms where the communication channel is constrained, costly or energy-consuming (e.g. underwater gliders). The SWE Bridge does not aim to take over or interfere with the navigation controller, so the observation platform's operator can decide when and how the data will be transmitted to the shore station.

The next step in the architecture is the Sensor Observation Service (SOS), a standardized service to archive data and metadata. It is compliant with the O&M model and provides a robust and semantically-tied way to archive sensor data and metadata. This service, typically located at the shore station, is not only used as

data archive, it is also used as gateway to provide access to both data and metadata. Web services, data visualization tools, data hubs or users may access the data using its API. Within this thesis the open-source implementation of the 52north company has been used [61], [63]. However, this component could be easily replaced by any O&M-compliant service, such as SensorThings API [64].

This universal architecture has been widely used within this thesis. It was firstly presented at the paper A.1 section 2. In the paper A.2 section II it is extended in order to facilitate the integration of hydrophones to provide real-time ocean sound data in an interoperable manner. The conference paper A.4 provides a use case of this architecture with the EGIM sensor system (see section 4.1). Finally, the conference paper A.3 provides a framework to not only acquire data, but also to provide a remote configuration service for (near)real-time configuration.

2.3.1 Sensor Deployment File

A Sensor Deployment File (SDF) is a SensorML file that describes a sensor, providing all the information required to be integrated into an observation platform. It is thoroughly described in the paper A.1 section 4. Its UML (unified model language) model is depicted in figure 2.5. A SDF is composed by two main components: the sensor instance and the sensor mission. The sensor instance contains the description of a sensor (name, serial number, communication protocol, etc.). It does not change over time and is re-usable between deployments. A sensor mission is the application of a sensor: where it is deployed, platform where the sensor is hosted, sensor configuration, acquisition workflow, etc. This section needs to be updated for every deployment.

Within the sensor instance, identifiers, contacts, communications interface and communication protocol are defined. Special attention must be put on the proper definition of the communication protocol, in order to provide a consistent protocol abstraction mechanism. By means of this abstraction, it is possible to describe virtually any existing communication protocol, whether standard or proprietary. Some communication protocols abstracted using an SDF file during this thesis are: NMEA standards (National Marine Electronics Association), SCPI (Standard Commands for Programmable Instruments) and RTP (real-time transmission protocol). More details can be found in chapter 4 and in papers A.1, A.2 and A.4.

The sensor mission includes deployment information and acquisition workflow details. This section is used by the SWE Bridge to set up the acquisition chain. More details on SWE Bridge's workflows are provided in section 2.4

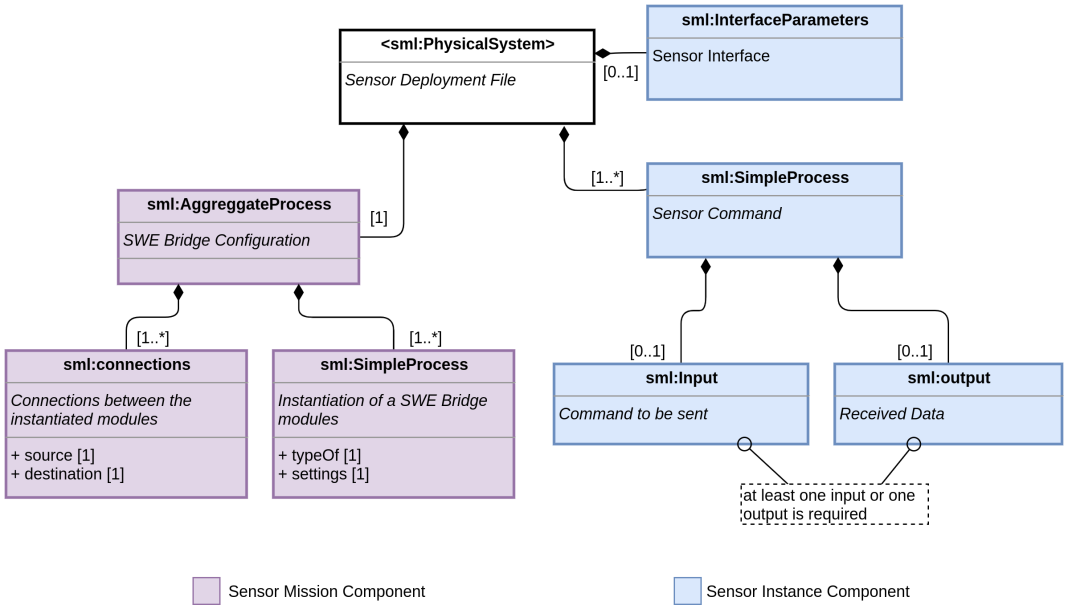


Figure 2.5: Sensor Deployment File UML diagram

2.3.2 Hydrophone SensorML Description

A Hydrophone SensorML Description (HSD) file is a particular case of SDF that contains the information required to interface a hydrophone. The UML model of a HSD is depicted in figure 2.6. Due to its particularities, hydrophones tend to be more complex than regular marine sensors. Thus, their low-level specifications need to be encoded (e.g. sensitivity, preamplifier gain, ADC, etc.). The HSD also provides the parameters to fine-tune the real-time signal processing, such as the frequency resolution, time window, window function to be applied, etc. The HSD template is thoroughly described in the paper A.2, section IV.

2.3.3 OGC PUCK and off-the-shelf Sensors

Within this work, sensors are classified into three groups: commercial off-the-shelf (COTS), OGC PUCK-enabled sensors and virtual instruments (VIs).

Off-the-shelf sensors are the majority of commercial sensors. They do not have any enhancement in terms of interoperability and generally use proprietary communication protocols. On the other hand, PUCK-enabled sensors are those which implement the OGC PUCK Protocol alongside their own proprietary protocol [56]. It is important to remark that the OGC PUCK protocol only defines a very limited set of commands that provide access to an internal memory, but it does not specify a common layer to configure and query sensors. Thus, OGC PUCK-enabled sensors still need to be modelled and abstracted using SDF/HSD descriptions.

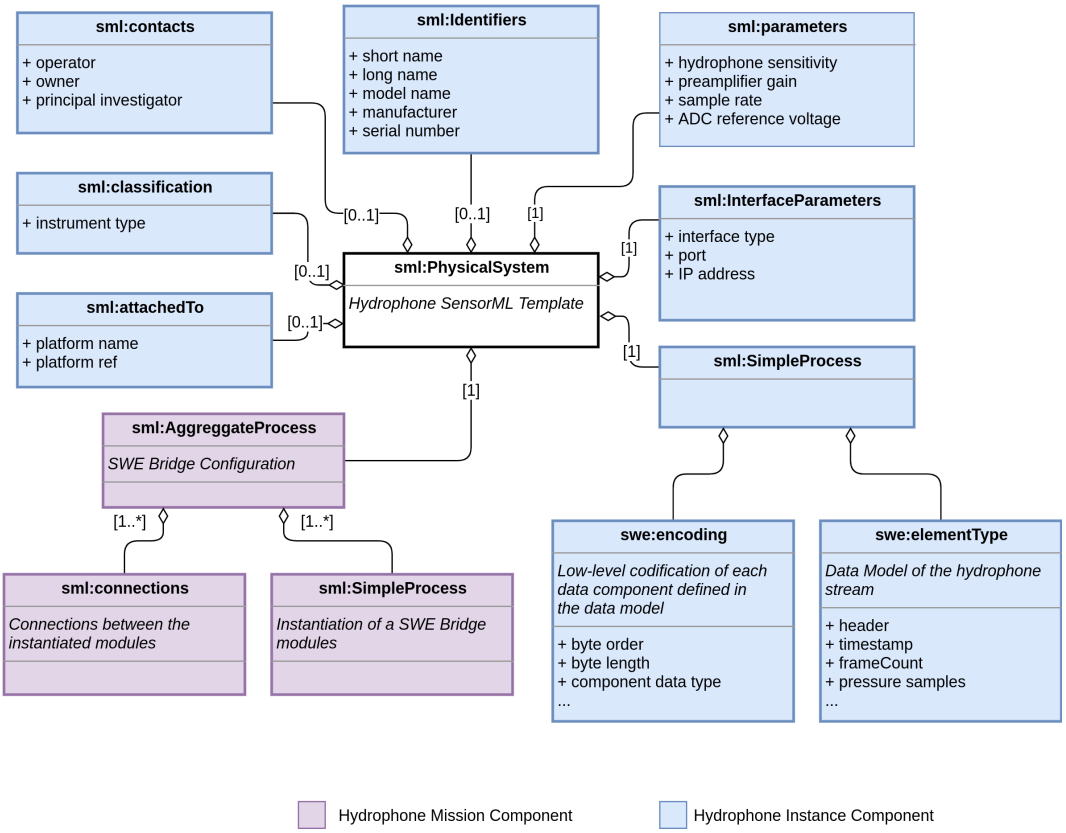


Figure 2.6: Hydrophone SensorML Description (HSD) UML diagram

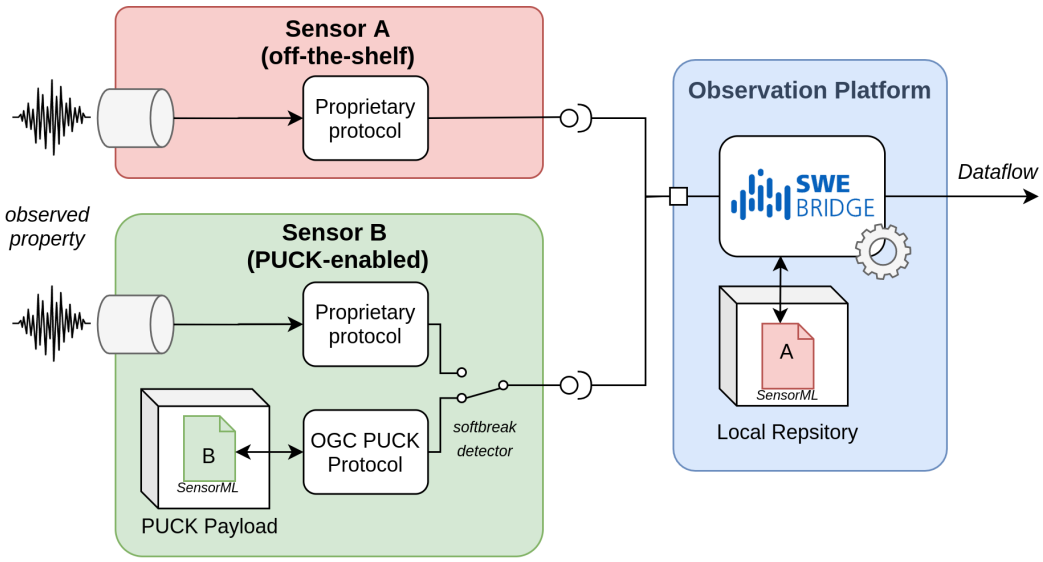


Figure 2.7: Integration strategies for PUCK-enabled and off-the-shelf instruments

Virtual Instruments (VIs) are software components that process, merge or generate data, making it available through a communications interface. Although they are

not physical components, from the integration point of view they are identical to off-the-shelf sensors since they provide data through an interface.

Figure 2.7 shows the sensor integration scheme of two sensors: an off-the-shelf sensor and an OGC PUCK-enabled sensor. Off-the-shelf sensors do not have any enhancement in terms of interoperability, thus their SensorML description should be stored in a local repository at the observation platform. When launching an instance of the SWE Bridge, a locally-stored SensorML file should be supplied. Within this file, the communications interface details should be stated (e.g. serial port /dev/ttyUSB1, baudrate 115200 bps, etc.). Then, the SWE Bridge can open the interface and communicate with the sensor.

On the contrary, when interfacing OGC PUCK-enabled sensors, the operator only needs to specify a port to the SWE Bridge. Then, it will try to detect any sensor plugged at that port by means of the OGC PUCK softbreak. Once it replies, the SWE Bridge will retrieve its SensorML description embedded within the payload, decode it and setup an acquisition accordingly.

2.3.4 Enhancing off-the-shelf Sensors

It is possible to provide OGC PUCK capabilities to enhance off-the-shelf instruments by means of an external device, such as the Smart Cable (figure 2.8). This device, developed by the Cyprus Subsea Consulting Services company, acts as a man-in-the-middle between the instrument and the observation platform. During most of the time it provides transparent communications. However, when a softbreak is detected, it takes control of the communications and gives access to its internal PUCK Payload using the OGC PUCK protocol. Thus, it provides OGC PUCK capabilities to any off-the-shelf instrument.

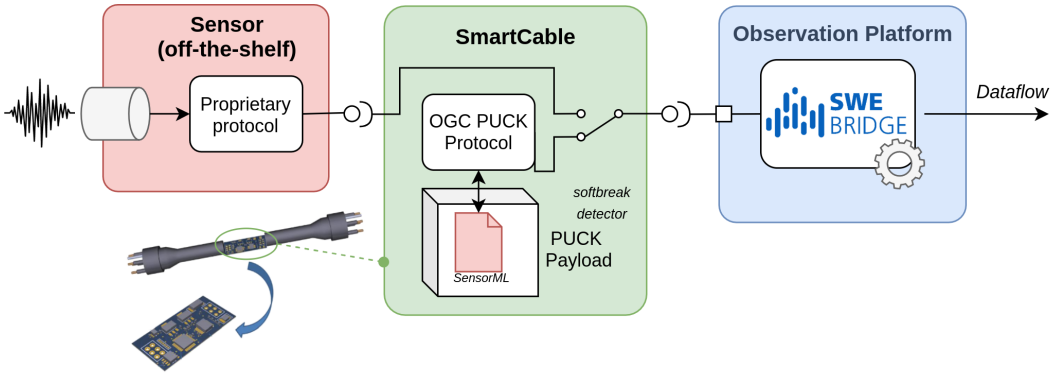


Figure 2.8: Integration scheme of an off-the-shelf instrument using a Smart Cable

2.3.5 SensorML and Semantic Interoperability

Generating a SensorML instrument description file containing all the desired information in a human-readable fashion is not always enough. Consider a SensorML description file describing a hydrophone. Within this file, it should be stated that the sensor can be classified as a hydrophone, such as the following SensorML excerpt:

Listing 2.1: SensorML hydrophone classifier example

```
<sml:classifier>
    <sml:Term>
        <sml:label>instrument type</sml:label>
        <sml:value>hydrophone</sml:value>
    </sml:Term>
</sml:classifier>
```

From the human point of view it is clear what "instrument type" and "hydrophone" means. However, for a machine this is not a trivial task. In fact from the machine point of view they are just strings, a bunch of meaningless chars.

In order to provide unambiguous and machine understandable information, the key concepts in the XML excerpt should be defined using Semantic Web technologies, such as controlled vocabularies and ontologies [65]. Controlled vocabularies and ontologies are representations of knowledge in formal languages. An important aspect of ontologies and controlled vocabularies is that they provide relations between concepts (e.g. a "hydrophone" is a type of "instrument", "sea water temperature" is a type of "temperature"). The overall goal of semantic web technologies is to provide machine-understandable information and context. In order to achieve this goal different technologies such as Resource Description Framework (RDF) and Uniform Resource Identifiers (URIs) are used.

The following excerpt contains the same information as listing 2.3.5, but with resolvable URIs pointing to machine-understandable controlled vocabularies with information about the meaning and context of "instrument type" and "hydrophone" terms:

Listing 2.2: SensorML hydrophone classifier example with controlled vocabularies

```
<sml:classifier>
    <!-- instrument type concept definition -->
    <sml:Term definition="http://vocab.nerc.ac.uk/collection/W06/current/CLSS0002/">
        <sml:label>instrument type</sml:label >
        <!-- hydrophone definition -->
        <sml:value>http://vocab.nerc.ac.uk/collection/L05/current/369/</sml:value>
    </sml:Term>
</sml:classifier>
```

In the paper A.2 section IV a list of the different vocabularies used within this thesis can be found. These vocabularies ensure the compatibility of the SensorML descriptions with Semantic Web technologies.

2.4 SWE Bridge

The SWE Bridge is an open-source, universal driver that aims to integrate any kind of scientific sensor into observation platforms and inject sensor data into spatial data infrastructures, as described in papers A.1 and A.2. It is able to decode SDF and HSD files and configure an acquisition chain accordingly, achieving on-the-fly sensor integration. Moreover, the SWE Bridge itself is described in SensorML, so humans and machines can access its representation in order to inspect its capabilities. For a more complete description of the SWE Bridge and its functionalities, its user manual can be found in appendix B.

The SWE Bridge implements the OGC PUCK protocol, so it can automatically detect and connect to OGC PUCK-enabled sensor. However, in the case of off-the-shelf sensors, the operator must specify the location of the appropriate SDF or HSD when a SWE Bridge instance is launched.

It has been designed bearing in mind the heterogeneity of state-of-the-art observation platforms, so it uses minimal resources, as detailed at A.1 section 5. It is coded using the C language and abstracting all the hardware and operating system specific calls by means of intermediate wrappers, achieving a portable code.

The SWE Bridge implements various interfaces such as serial (UART, RS232, RS485, etc) and Ethernet (TCP/IP, UDP) communications.

It has a modular design, separating common acquisition functionalities within a set of modules. These modules can be instantiated and arranged in order to set up acquisition chains, from simple "take a measurement, store it in a file" up to complex workflows with dozens of interconnected operations. In A.1 section 6 there are some examples of complex workflows configured using the SWE Bridge. The modules implemented in the SWE Bridge are detailed in section 2.4.3.

The code is available at the following public git repository.

2.4.1 Operation

The SWE Bridge operation is organized in four components that are executed sequentially: the OGC PUCK Detector & Extractor, the decoder, the SensorML Interpreter and the Mission Scheduler. The execution of the SWE Bridge may vary slightly depending on whether the sensor is OGC PUCK-enabled or not, as shown in figure 2.9. OGC PUCK-enabled sensors shall provide their Sensor Deployment File (SDF, see section 2.3.1) embedded within their payload memory. However, when interfacing a commercial off-the-shelf sensor (COTS) its associated SDF shall be passed as argument to the SWE Bridge.

The OGC PUCK Detector and Extractor detects new OGC PUCK-enabled

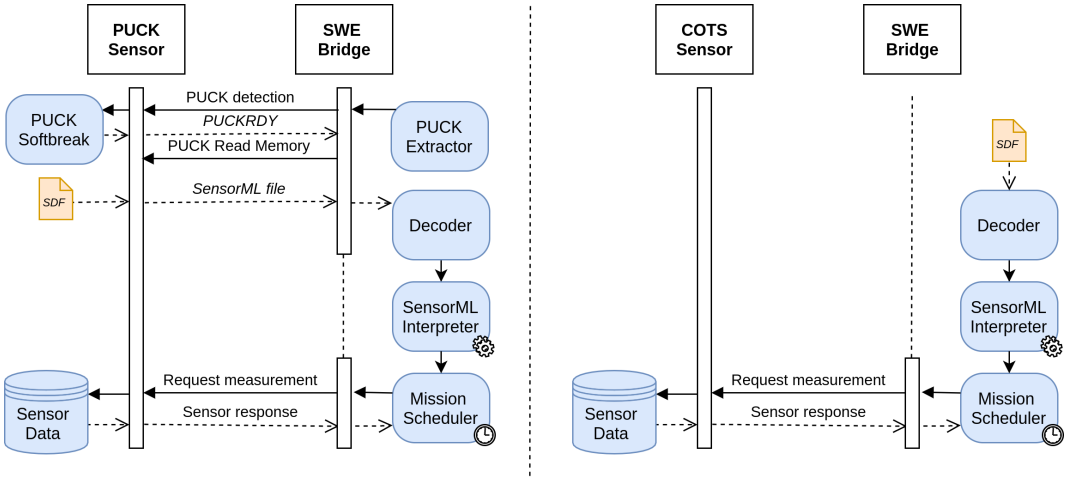


Figure 2.9: SWE Bridge execution diagram for PUCK-enabled and commercial off-the-shelf sensors (COTS)

sensors connected to the communications interface. Once a new sensor is detected, this component extracts its SDF. When interfacing a COTS sensor, a local SDF can be passed as argument to the SWE Bridge, bypassing the OGC PUCK Detector and Extractor component.

The decoder extracts the desired information from the SDF, which can be encoded in plain XML or in the efficient EXI format. The SensorML Interpreter service takes the extracted information and uses this data to configure the data acquisition workflow. If not configured by the PUCK extractor, the communication interface is also setup according to the information decoded from the sensor description. Afterwards, using the information from the sensor mission, the scheduler executes the data acquisition workflow, managing the sensor and storing sensor data.

2.4.2 Hardware Resources

The SWE Bridge aims to be integrated into any platform, regardless of its underlying hardware resources. In order to achieve a cross-platform implementation, the SWE Bridge has been implemented using ANSI C, with special emphasis on minimizing the usage of underlying software and hardware resources. All platform-dependent resources are abstracted using resource abstraction wrappers, which provide a unified way to access the platform's resources (see figure 2.10). These wrappers are the only functions that need to be adapted when deploying the SWE Bridge in a new platform.

The boxes with dashed lines represent the optional parameters that can be implemented, depending on the observation platform. In example, an observation platform that is deployed in a fix position does not require to implement the GPS wrappers.

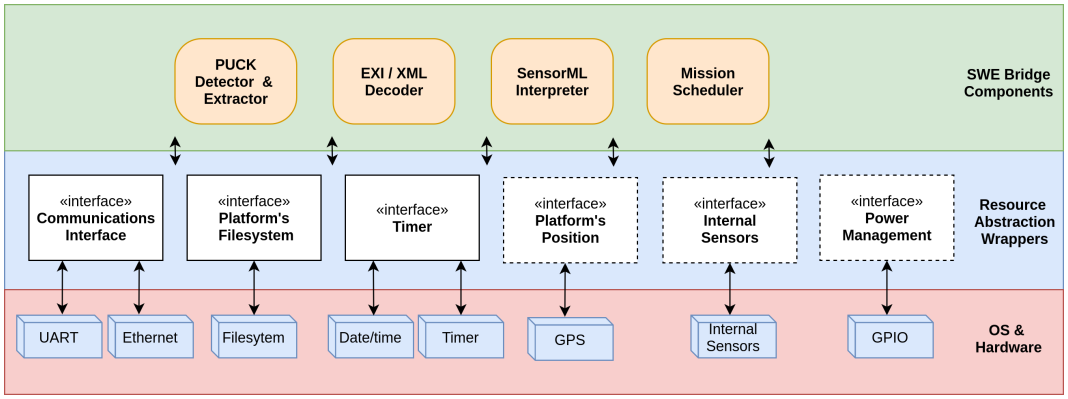


Figure 2.10: Sensor Deployment File overview

2.4.3 Modules

SWE Bridge Modules are built-in implementations of generic operations. Each module implements a generic operation, such as send a command, store data to a file, power on/off a sensor, etc. These modules can be instantiated within the SWE Bridge to generate processes, which can be connected generating a workflow. The term "process" is used when a module is called and loaded into the SWE Bridge's memory. As an analogy with object-oriented programming languages, "module" is a "class" while a "process" is an "object". The following modules are implemented within the SWE Bridge:

1. **Instrument Command**: Sends and/or receives a sensor command
2. **Hydrophone Stream**: receives and process high frequency streams from a hydrophone
3. **Insert Result**: Stores sensor data into an insert result file (O&M data)
4. **CSV Generator**: Generates CSV files from sensor data
5. **Linear Calibration**: Applies linear calibration to sensor data
6. **Subsampling**: Subsamples incoming data
7. **Power Management**: Powers on/off sensors (requires GPIO hardware)
8. **Internal Sensors**: Reads platform's internal sensors (requires hardware internal sensors)
9. **Analogue Measurement**: Performs an analogue measurements (requires ADC hardware)

10. **Zabbix Sender**: Sends data to a Zabbix monitoring service (requires external zabbix sender application)
11. **Sound Pressure Level**: Calculates SPL, RMS and SEL levels from a hydrophone stream or WAV file
12. **WAV Generator**: Generates WAV files from a hydrophone stream
13. **WAV Reader**: Reads acoustic data from WAV files

2.4.4 Generating Acquisition Workflows

To generate a workflow using the SWE Bridge the first step is to instantiate a set of modules. When a module is instantiated, a process is generated and loaded into the SWE Bridge Scheduler. These processes can be connected in a process chain, generating complex workflows.

The processes in a workflow pass data from source to destination (left to right in the diagrams in this section). Some processes generate data (e.g. instrument command), others require previous data to operate correctly (e.g. insert result) and some are even data-independent (power management).

Although there are no restrictions in which processes can be connected, some configuration may not be valid. For instance, if it is not possible to connect a data-less process (e.g. power on/off sensor) to a process that requires input data (e.g. insert result). However, it is possible to connect processes that generate data to a process that does not require data (input data will be ignored).

In figure 2.11 a simple workflow is depicted. Two modules are instantiated, generating the "takeSample" process, and the "storeResult" process. The takeSample is an instrument command that periodically queries the sensor for data, and insert result stores this data into O&M files.

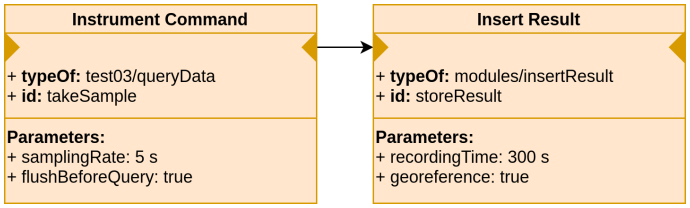


Figure 2.11: Example of a simple workflow

The previous workflow is very simple and only has two components, but using SWE Bridge module's much more complex workflows can be created. In figure 2.12 a complex workflow with 6 processes is depicted. This example models the scenario where the SWE Bridge periodically powers on the sensor, takes some measurements,

and then powers it off again. Data gathered by the instrument command processed is stored in CSV format and in O&M format. Note that previously to the insert result process, the data is subsampled. Thus, data in O&M format will be a subsampled dataset, while CSV data will be the full dataset.

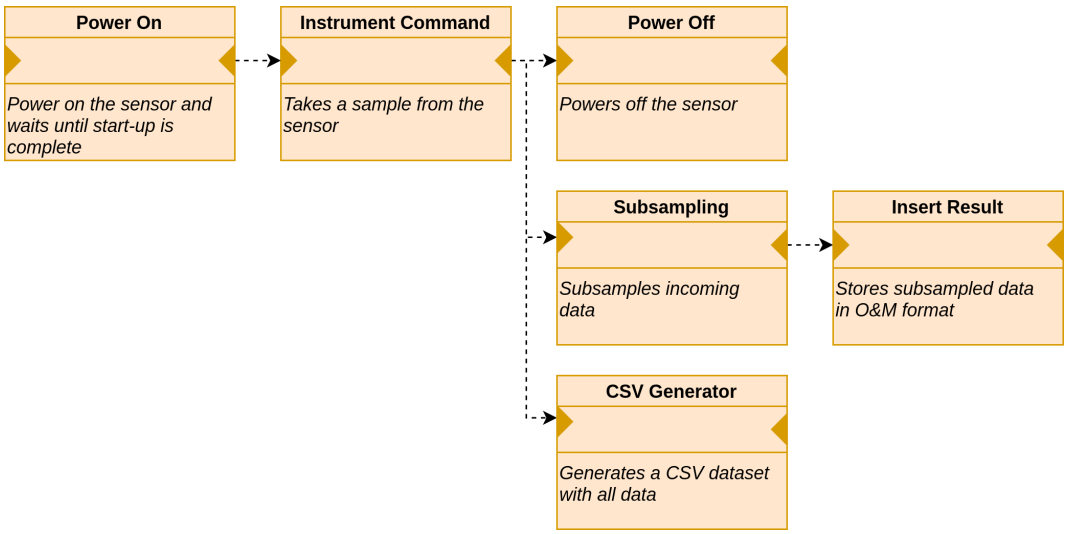


Figure 2.12: Example of a complex workflow

2.4.5 SWE Bridge Model

The SWE Bridge itself is also described using the SensorML standard, providing syntactically-tied description of its functionalities. Every SWE Bridge module is encoded within a `sml:SimpleProcess`, containing a generic description of the module, inheritances, the module identifier and a set of parameters.

Each parameter also has its own description and identifiers. Furthermore, it uses the SWE Common Data model formats to specify the type of data used to configure the module's parameter. The following example shows how the Instrument Command is described:

Listing 2.3: Instrument command definition within the SWE Brdige's model

```
<sml:component name="instrumentCommand">
    <sml:SimpleProcess>
        <gml:description>
            Module to send and/or receive commands from an instrument. It is required to
            have at least one input and/or one output. The input encodes the command to
            be sent to the instrument while output describes the expected response. In
            the case of streaming instruments only an output without input needs to be
            defined. In the case of commands without response, only an input needs to be
            defined.
        </gml:description>
        <gml:identifier codeSpace="swebridge">swebridge:instrumentCommand</gml:identifier>
        <sml:typeOf xlink:href="swebridge:modules:genericComponent"/>
```

```
        <sml:parameters>
            <sml:ParameterList>
                <!-- timeOut -->
                <sml:parameter name="timeOut">
                    <swe:Quantity optional="false" id="timeOut">
                        <swe:description>
                            The module will return error if a response is not received
                            before the timeout expires.
                        </swe:description>
                        <swe:uom code="seconds"/>
                        <swe:value>2</swe:value>
                    </swe:Quantity>
                </sml:parameter>
                ...
            </sml:ParameterList>
        </sml:parameters>
    </sml:SimpleProcess>
</sml:component>
```

In the example a simplified description of the instrument command module is presented. The first element in teh excerpt is a human-readable description of the module. An identifier is defined and it inherits properties from the generic module (abstract class containing common properties for all modules). Additionally, the `timeout` parameter is also defined as a Quantity (floating point value), using seconds as unit of measurement. The timeout parameter is mandatory (optional attribute is set to False) and it has a value of 2 seconds by default.

The SWE Bridge Model is the building block to achieve interoperability in any future SDF/HSD editor. Since all the information is structured, it can be downloaded and easily interpreted by any software component. Then, this information can be displayed using graphical interface, where the user can select and configure the modules to create SDF files.

2.5 Conclusions

Within this chapter an interoperable architecture to achieve plug-and-play sensor integration has been proposed. The requirements to integrate sensors into observation platforms and its data into spatial data infrastructures have been analyzed. Then, existing standards and protocols able to fulfill these requirements have been discussed: OGC PUCK to embed sensor metadata within the sensors; SensorML to abstract sensor metadata; O&M to provide an unambiguous model for the acquired data; SOS to provide data/metadata archive and access; and finally EXI to compact the information to be shared among nodes in systems with limited resources.

One of the key features of the proposed architecture is a sensor abstraction mechanism based on the SensorML: the SDF (for generic sensors) and the HSD (SDF extended with hydrophone-specific information). These metadata templates

include in a machine-understandable manner all the required information to integrate sensors on-the-fly with standard-based data services. Thus, the thesis objective 1 is accomplished with the SDF and HSD standardized metadata models. In order to embed these sensor metadata descriptions within the sensors themselves, the OGC PUCK Protocol is discussed.

Then, SWE Bridge is presented: a tool that is able to interface sensors on-the-fly based on their standard SDF/HSD descriptions. It has many features and functionalities, such as universal data acquisition, real-time data processing, data harmonization and more. The SWE Bridge provides standardized outputs, compatible with standard data services such as SOS. Its implementation is carefully decoupled from any particular hardware or operating system, making compatible with almost any platform (desktop computers, single-board-computers, microcontrollers, etc.). Furthermore, the SWE Bridge itself has an extensive machine-understandable description based on the SensorML standard. This description can be used by other processes to understand its functionalities and capabilities, laying the foundations for automated SDF/HSD generation, data provenance and data traceability.

Combining the presented elements a universal end-to-end data acquisition architecture is presented, from the sensor to spatial data infrastructures. This architecture builds on existing standards and ensuring data/metadata interoperability. Furthermore, its focus on open-source implementations with a strong focus on re-usability. Thus, the presented architecture effectively fulfills the thesis objective 2.

Chapter 3

Ocean Sound Monitoring

Ocean sound measurements are sometimes difficult to compare because different measurement methodologies or acoustic metrics are used, leading to a risk of misunderstandings between scientists from different disciplines [19]. Underwater sound is usually measured using the Sound Pressure Levels (SPL) over a time window T at different third-octave band frequencies [19]. However, the time window and the frequency bands may be adjusted by the operator depending on the deployment.

In this section an algorithm to obtain sound pressure level measurements is discussed. The proposed algorithm follows best practices on the field and uses community-accepted procedures [10, 11, 19, 66]. Furthermore, this work focuses on providing an implementation that can be seamlessly integrated into observation platforms with constrained resources to provide real-time, *in situ* measurements. One of the critical aspects of these platforms is the computational cost required to apply underwater sound level algorithms in real-time, while maintaining a reasonable frequency resolution (Δf).

The proposed algorithm and its implementation has to be flexible enough to be able to interface with any existing hydrophone, regardless of its characteristics. Moreover, it has to be sufficiently generic to allow users to tune its parameters to fit in different applications. Some of these parameters are the time window, frequency resolution or the band frquencies analysed.

3.1 Hydrophone Acquisition Chain Modelling

Generally, hydrophones provide raw data (dimensionless samples). Thus, the first step to achieve a generic algorithm implementation is to convert from raw dimensionless samples to pressure samples in μPa. To achieve this conversion, the acquisition chain of the hydrophone has to be modelled. Hydrophones are complex instruments that

include several components in their acquisition chain.

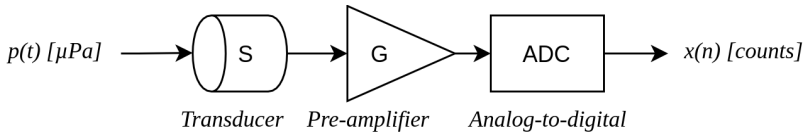


Figure 3.1: Typical signal conditioning stage of a hydrophone

The typical acquisition chain of a hydrophone (depicted in figure 3.1) includes a transducer, a preamplifier and an analogue-to-digital converter (ADC). In the paper A.2 section III-A a detailed analysis on how to convert from ADC samples into a pressure signal is discussed, alongside with all the characteristics that are required (hydrophone sensitivity, ADC reference voltage, number of bits, etc.). The equations from the paper are replicated in this section to enhance the readability of the thesis.

In order to convert from raw samples to pressure signal the following hydrophone parameters need to be known:

- S : transducer sensitivity (dB re 1 V/μPa)
- G: pre-amplifier gain (dB)
- V_{ref}: ADC's reference voltage (V)
- M: ADC's resolution in bits (dimensionless)
- f_s: Sample Rate (Hz)

The conversion from raw samples to pressure signal is modelled with equations (3.1) and (3.2).

$$p(n) = \frac{x(n)}{S_{lin} G_{lin} \frac{1}{LSB}} [\mu Pa] \tag{3.1}$$

Where $p(n)$ is the discrete pressure signal, $x(n)$ is the raw digital counts provided by the ADC and LSB^{-1} (least significant bit) is the ADC conversion coefficient in $counts/V$. Note that S and G have been converted from dB to linear, S_{lin} and G_{lin}.

Then , the LSB can be calculated from the ADC's specifications using (3.2):

$$LSB = \frac{V_{fs}}{2^M} = \frac{2V_{ref}}{2^M} \; [V/counts] \tag{3.2}$$

Where M is the ADC's number of bits, V_{fs} is the ADC's full-scale voltage range and V_{ref} is ADC's reference voltage. In (3.2), it has been assumed that the hydrophone's ADC has a symmetric reference voltage ($V_{ref+} = -V_{ref-}$).

3.2 Sound Pressure Level

Ocean sound is usually characterized by the Sound Pressure Level (SPL). The SPL is a logarithmic variable that provides an average of the pressure signal over a time integral. The overall SPL value over a time window T can be calculated using equation (3.3) [19, 66]. It is expressed in dB re μPa.

$$SPL_N = 10 \log_{10} \left(\frac{1}{N} \sum_N \frac{p(n)^2}{p_0^2} \right) [dB\ re\ \mu Pa] \tag{3.3}$$

Where N is the number of samples within the time window ($N = T \cdot f_s$) in the pressure signal's segment and the reference pressure in seawater is $p_0 = 1$ µPa.

Since SPL values are an average over several pressure samples, the resulting value is highly dependent on the time window T [19]. Thus, when providing SPL values, it is very important to clearly state the time window used.

The calculated SPL levels are highly dependent on the hydrophone's bandwidth. Thus, in order to achieve levels comparable through different deployments, the MSFD states that the SPL levels have to be computed over third-octave band frequencies [9, 10]. However, obtaining the energy within band frequency is slightly more complicated.

In addition to the SPL values, other parameters may be of interest when analysing ocean sound, such as the Sound Exposure Level (SEL) or the Root Mean Square (RMS). These parameters are discussed in the paper A.2 section III-C.

3.3 SPL Algorithm Comparison

Within this thesis, an efficient algorithm with minimum usage of computational resources is envisioned. This efficiency is crucial to embed the algorithm in resource-constrained platforms, such as underwater gliders or autonomous dataloggers. These platforms have limited power availability and communication's bandwidth. Thus, an efficient algorithm that requires less computational power will reduce energy consumption and effectively increase the platform's autonomy.

Three different approaches where analysed to efficiently obtain SPL values over a frequency band: a filter bank, an FFT-based algorithm and the Goertzel's Algorithm. This comparison can be found in A.2, appendix A.

3.3.1 Filter Bank

The filter bank approach to calculate the third-octave SPL levels involves decimating and filtering, as shown in figure 3.2. Generally, hydrophones have a broad bandwidth,

up to tens or hundreds of kHz. Thus, its sampling rate is usually much higher than the frequencies of interest. In order to save computational resources and to ensure the filter stability, the incoming signal needs to be decimated.

The decimation process involves applying a low-pass filter to minimize aliasing, and down-sampling. SPL values are usually calculated for low-frequency third-octave bands. Thus, the incoming signal may be over-sampled by several orders of magnitude over the Nyquist rate. In this case, decimating the signal in a single stage may result in a very costly and complex filtering arrangement, so it may be more efficient to decimate the incoming signal in multiple stages [67]. The optimal number of decimation stages needs to be calculated at run-time based on the hydrophone's sampling rate and the highest cut-off frequency of the third-octaves bands. Obtaining the optimal amount of decimation stages is not a trivial task, since it depends on the ratio between the maximum frequency of interest and the sampling rate. Moreover, a slight modification in the maximum frequency of interest, may result in a huge difference in the decimation arrangement.

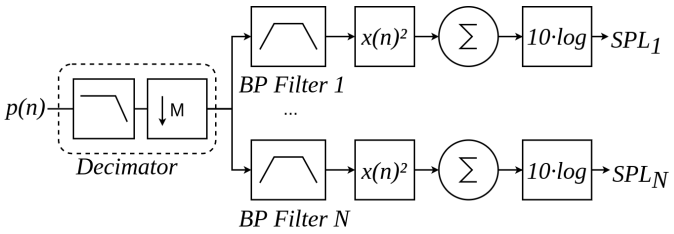


Figure 3.2: Filter Bank algorithm block diagram

Once the signal has been decimated, only the energy within a specific band-frequency needs to be taken into account. In order to select only a specific band a band-pass filter is used. Ideally such a filter would completely eliminate the energy in the rejection-band without affecting the power in the pass-band. In practice, it should have small ripple at the pass-band and a fast roll-off in the rejection bands to minimize its influence on the signal. When designing a filter there is a trade-off between the computational cost and the filter's performance. The band-pass filter used within this work was a 5th order elliptic filter with passband ripple of 1 dB and rejection band of -80 dB. A filter is calculated at run-time for each third-octave band used in the analysis, as shown in figure 3.3.

3.3.2 Fast Fourier Transform

The Fast Fourier Transform (FFT) is a well-known signal processing technique to obtain the Digital Fourier Transform (DFT) in a computationally-efficient manner for data segments with power-of-two length. The vast majority of processing software

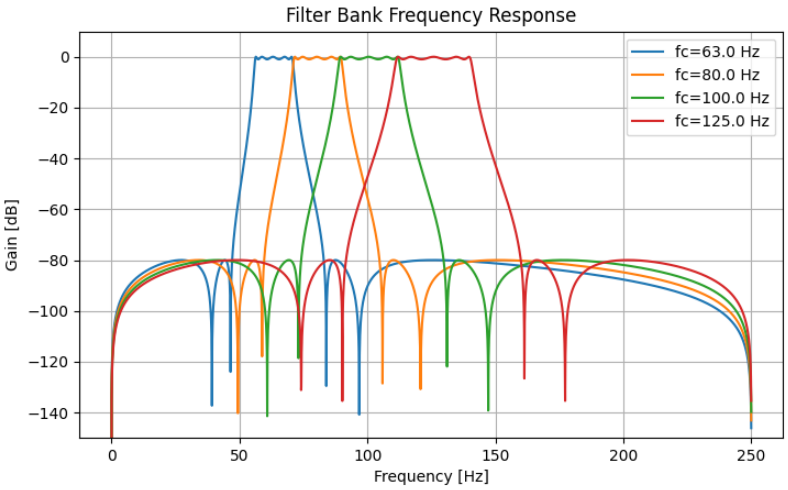


Figure 3.3: Frequency response of a filter bank composed by 5th order bandpass elliptic filters calculated using python3's scipy library. Passband ripple is set to 1 dB and the rejection band is set to -80 dB

implement the FFT as its primary algorithm to compute the spectrum of a signal.

In order to obtain the SPL values over a specific third-octave band, the FFT can be used to obtain the periodogram of the signal. The periodogram is an estimate of the Power Spectral Density (PSD) of the pressure signal, which can be used to obtain the SPL over a specific frequency band. This is thoroughly analysed in the paper A.2, section III. A block diagram of the FFT-based algorithm for ocean sound is depicted in figure 3.4.

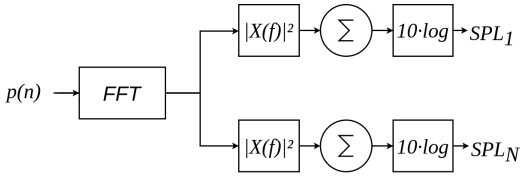


Figure 3.4: Ocean sound algorithm block diagram based on the Fast Fourier Transform (FFT)

3.3.3 Goertzel's Algorithm

Similarly to the FFT, the Goertzel's algorithm is an implementation of the DFT. Its block diagram is depicted in figure 3.5. It is much slower than an FFT, but it has the ability to independently calculate DFT terms. The Goertzel algorithm has been extensively used in embedded digital signal processing applications, such as the recognition of dual-tone multi-frequency signaling used in analog telephones.

According to the literature, the Goertzel's algorithm can present computational savings when only a few DFT bins M are computed, i.e. $M < log_2 N$ [68]. When

calculating SPL values over a specific frequency band, only few frequency bins are of interest. Thus, it has been considered in the comparison. A block diagram of the Goertzel's algorithm for ocean sound is depicted in figure 3.4.

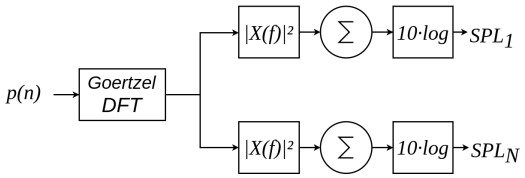


Figure 3.5: Ocean sound algorithm block diagram based on the Goertzel's algorithm.

3.3.4 Comparison Result

The algorithms were implemented and compared to evaluate their performance. Two characteristics where evaluated, the amount of memory used (RAM) and the execution time. The algorithms where tested in two different conditions: first increasing the number of third-octave frequency bands to be calculated (with a fixed number of samples), and then increasing the number of samples (with a fixed number of third-octave frequency bands).

The implementation was done using the python3 programming language. Although it is a high-level scripted programming language and most of the complex operations are done under-the-hood, it is a good starting point to assess the feasibility of each algorithm and have a rough estimate of their computational cost. In figure 3.6 the comparison results are depicted. A more detailed analysis of the results is provided in the paper A.2, appendix A.

The filter bank has a worse performance than the FFT in terms of execution time. Additionally, its memory usage increases as more bandpass filters are applied. The Goertzel's algorithm is only useful when very few frequency bins are calculated, but the resulting frequency resolution is not acceptable (tens of Hz), When the frequency resolution is reduced its execution time grows exponentially, showing a very poor performance. The FFT algorithm outperforms both Goertzel's algorithm and filter-bank in execution time, additionally it has a very stable and predictable memory usage. Thus, it has been selected and implemented within the SWE Bridge.

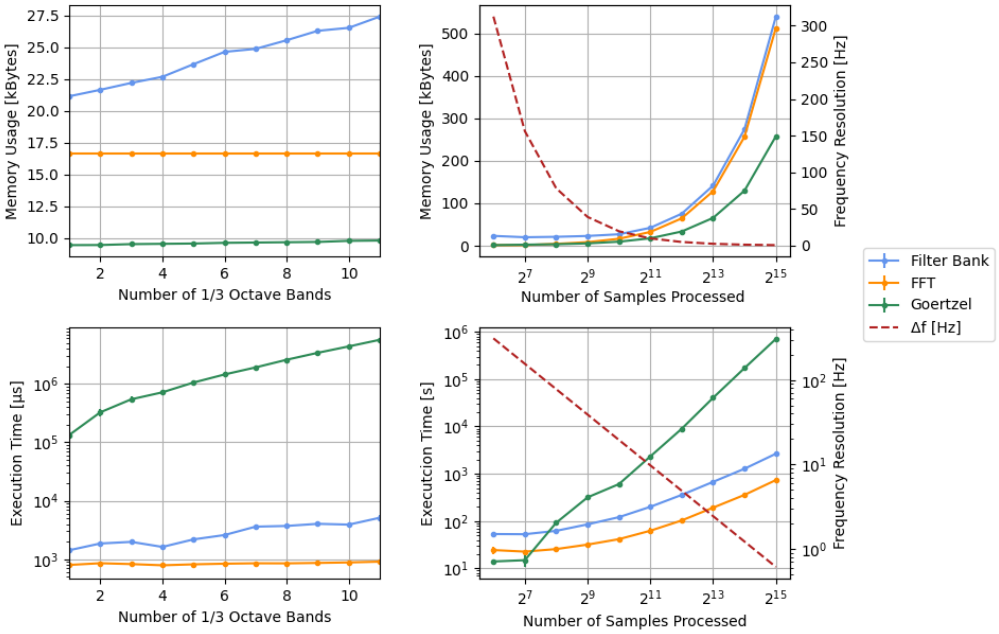


Figure 3.6: Performance test of three different SPL algorithms based on their third-octave estimation method: filter bank, FFT and Goertzel's algorithm (source: paper A.2). The left graphs show the memory usage (top) and the execution time (bottom) depending on the number of third-octave bands calculated, starting from the band centered at 63 Hz (N is set to 4096). The right graphs show the memory usage (top) and execution time (bottom), depending on the number of samples in the time window (only 63 and 125 Hz third-octave bands). The dashed red line in rights graphs represent the frequency resolution Δf of the DFT for each value of N. The sampling frequency is set to 20 kHz.

3.4 SPL Algorithm Implementation

An algorithm to calculate SPL, SEL and RMS has been integrated into the SWE Bridge universal driver. Using HSD descriptions (see section 2.3.2) the characteristics of the hydrophone's acquisition chain are modelled and abstracted, interfacing it and acquiring the pressure signal. In the paper A.2, section III-B a detailed analysis on how to calculate the SPL, SEL and RMS over third-octave bands is presented. The equations from the paper are replicated in this section to enhance the readability of the thesis.

3.4.1 Third-octave Frequency Bands

Based on the algorithm comparison's results, the FFT-based algorithm to obtain SPL at arbitrary frequency bands is implemented. The periodogram is contemplated as a tool to estimate the Power Spectral Density (PSD) of the pressure signal over a time

window, as defined in (3.4).

$$P_{xx}(f) = \frac{1}{N} |P(f)|^2 \, [\mu Pa^2/Hz] \tag{3.4}$$

Where P_{xx} is the periodogram over a time interval $T = N/fs$, $P(f)$ is the Discrete Fourier Transform (DFT) of the discrete pressure signal $p(n)$ and f is the normalized frequency [68].

In order to reduce the spectral leakage introduced by the finite length of the data segment in real-world applications, it is possible to compute the periodogram of the windowed pressure as defined in 3.5 (also named modified periodogram) [69].

$$p_w(n) = w(n) \cdot p(n) \tag{3.5}$$

Where $w(n)$ is a window function of length N samples and $p_w(n)$ is the windowed pressure signal. Applying a window function reduces the spectral leakage, but it has severe implications on the resulting spectrum [69]. In order to correct these undesired side-effects, the result of the DFT has to be scaled with the coherent gain (CG). This factor can be calculated as the sum of the window components (W) normalized by the number of samples (N), as showed in (3.6) and (3.7).

$$W = \sum_N w(n) \tag{3.6}$$

$$CG = \frac{W}{N} \tag{3.7}$$

From the spectral point of view, the amount of energy within each DFT bin is also modified by the window function. Each DFT bin can be understood as a very narrow band-pass filter with a Δf bandwidth. However, when a window is applied, the bandwidth of this hypothetical filter is slightly increased due to the aperture of the window's spectral response main lobe [70]. To correct the extra power contribution in each DFT bin due to the bandwidth increment, the normalized equivalent noise bandwidth ($nenbw$) correction factor is used. The $nenbw$ factor can be calculated using (3.8) [70].

$$nenbw = N \frac{\sum_N w(n)^2}{W^2} \tag{3.8}$$

In order to compensate the mentioned side-effects of windowing, the periodogram of a windowed signal (also known as modified periodogram) can be calculated using

(3.9) [68].

$$P'_{xx}(f) = \frac{1}{N \cdot nenbw} \left| \frac{P_w(f)}{CG} \right|^2 = \frac{|P_w(f)|^2}{\sum_N w(n)^2} \tag{3.9}$$

Being $P_w(f)$ the DFT transform of the windowed pressure signal $p_w(n)$.

Due to its high variance, the periodogram provides a rough estimate of the PSD. In order to reduce this variance and improve the PSD estimate, the Welch and Bartlett's methods are implemented [71]. These methods average several periodograms to reduce the variance of the PSD estimate and improve the execution time at the expense of frequency resolution. The Bartlett's method uses sequential non-windowed segments while the Welch's method uses windowed overlapped data segments. Both methods can be applied using (3.10).

$$\overline{P}_{xx}(f) = \frac{1}{K} \sum_{i=0}^{i=K-1} P'_{xx_i}(f) \tag{3.10}$$

Where $\overline{P}_{xx}(f)$ is the power spectral density estimation, K is the number of periodograms being averaged and $P'_{xx_i}(f)$ is the periodogram of the ith data segment. The number of segments to be averaged depends on the length of each data segment N and on the overlapping [71].

Then, the SPL values over an arbitrary frequency band can be calculated using (3.11) [72].

$$SPL_{f,N} = 10 \log_{10} \left(\frac{2}{N} \sum_{f_l}^{f_h} \overline{P}_{xx}(f) \right) [dB\ re\ \mu Pa] \tag{3.11}$$

Being f_l and f_h the low / high limit frequencies of the desired frequency band and N is the number of samples within each FFT segment. Since the PSD of a real signal is symmetric, the negative frequencies are redundant and can be omitted [70]. However, to consider their energy contribution, a factor of 2 has to be applied to the sum of the positive frequency bins. Note that the result of (3.11) depends on multiple parameters, including the total number of samples M, the number of samples in each data segment N, the overlapping between data segments and the window function.

Finally, a small error due to the finite number of frequency bands is expected. To compensate the error introduced by the difference between the theoreteical and practical bandwidth, equation (3.12) can be applied.

$$SPL'_{f,N} = SPL_{f,N} - 10 \log_{10} \frac{BW_{real}}{BW_{ideal}} \tag{3.12}$$

Where $SPL'_{f,N}$ is the sound pressure level value with the bandwidth correction,

BW_{ideal} is the ideal bandwidth of the third octave band (f_h - f_l) and BW_{real} is the real bandwidth summed in (3.11).

Additionally, the Sound Exposure Level (SEL) and the Root Mean Square (RMS) values can be calculated using equations (3.13) and (3.14).

$$SEL_T = SPL'_{f,N} + 10\log_{10}\left(\frac{T'}{T_0}\right)[dB\ 1\ \mu Pa^2 s] \tag{3.13}$$

Being T' the time window over which the SPL value was calculated (proportional to M) and $T_0 = 1$ second is the time reference.

$$P_N = \sqrt{\frac{1}{N}\sum_N p(n)^2}[Pa] \tag{3.14}$$

In order to maintain the same time window as SPL and SEL values, several RMS values are averaged, as stated in (3.15).

$$\overline{P}_{N,M} = \frac{1}{M}\sum_M P_N[Pa] \tag{3.15}$$

Being $\overline{P}_{N,M}$ the mean of M RMS values. The SWE Bridge provides the averaged mean of several RMS values for two reasons: to maintain the same time window used in SPL and SEL calculations, and to prevent the use of very large buffer which may result in memory overflow and excessive computational cost

3.4.2 Missing Packets and Timestamping

Hydrophones usually stream their samples grouped in packets using UDP communications. Due to the nature of these communications, some packets may be lost during the transmission. Thus, it is important to keep track of the missing packets to maintain the time-base and avoid glitches. Moreover, timestamping packets upon arrival is not trivial, since some hydrophones may include the timestamp in their headers, while others do not. In the paper A.2 section III-C a detailed description on how the SWE Bridge manages missing packets and timestamping is provided.

3.4.3 Acoustic Recordings

Although the main goal of this thesis is to provide a tool for in situ, real-time processing of ocean sound data, acoustic recordings may also be of interest. Recordings may be used to re-process data as well as extract other information (e.g. bioacoustics). Thus, the SWE Bridge includes the option to generate acoustic recordings in WAV files, whether continuously or using a duty cycle to save storage space.

The WAV format does not directly support metadata. However, as a workaround the ID3 informal standard is used within the SWE Bridge to embed hydrophone's metadata within WAV files. In the paper A.2 section IV-C the use of this informal standard is discussed.

3.5 SPL Algorithm Validation

The results produced by the underwater sound level algorithm integrated within the SWE Bridge have been compared and validated with the PAMguard software.

3.5.1 PAMguard

PAMguard is a popular desktop software application for semi-automated Passive Acoustic Monitoring (PAM) with special emphasis on cetaceans, widely used in the acoustics community [73]. It includes multiple sound processing algorithms, including cetacean detection, underwater sound levels and more.

PAMguard's built-in "Band Noise Monitor" module calculates underwater sound level monitoring at third-octave band frequencies based on a filter bank (see section 3.3.1). This module produces a set of SPL values compliant with the ANSI Standard ANSI S1.11-2004 American National Standard Specification for Octave-Band and Fractional-Octave-Band Analog and Digital Filters [74].

The hydrophones calibration parameters have been introduced in PAMguard and a set of third-octave filters have been generated using the Band Noise Monitor module. In figure 3.7 the PAMguard's Band Noise Monitor configuration is depicted, including the filter bank frequency response.

3.5.2 Validation Setup

Acoustic data acquired by an icListen hydrophone at PLOCAN has been processed by the SWE Bridge and PAMguard. This data has already been analysed and published in paper A.2, section V-B.

PAMguard was configured using 7th order Butterworh third-octave band filters with center frequency from 10 Hz up to 2000 Hz. The time window for SPL calculations was set to 10 seconds.

The SWE Bridge used the Welch's method, using a Hann window with 50 % overlap. The time window was set to 10 seconds and the frequency resolution of 1 Hz.

An hour of acoustic data was processed and its resulting datasets compared.

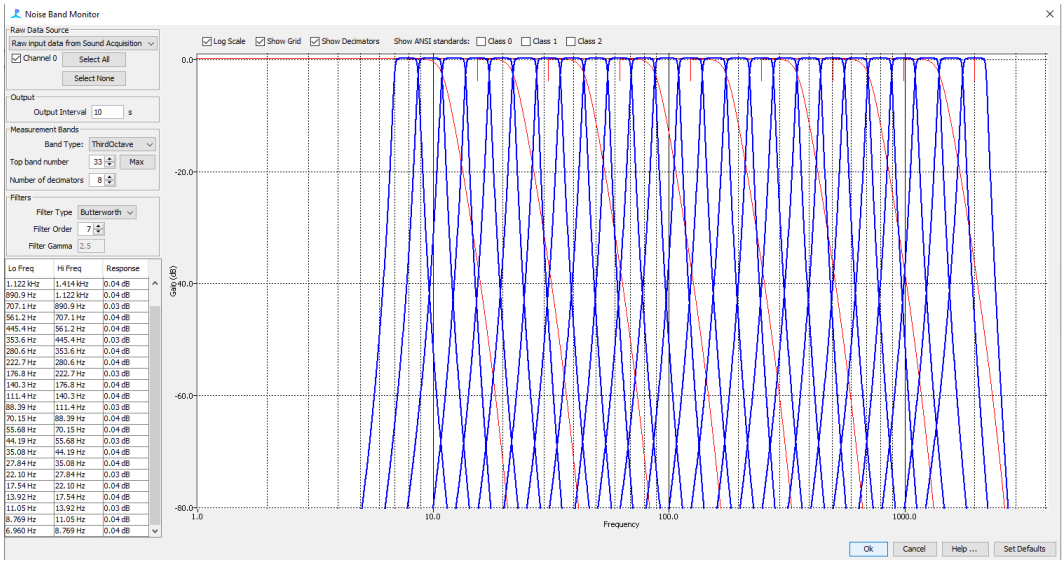


Figure 3.7: PAMguard Band Noise Monitor configuration displaying the filter bank frequency response.

3.5.3 Validation Results

The results generated by both algorithms are very similar, as shown in the timeseries of both algorithms depicted in figure 3.8. There is a significant difference at the first measurement caused by filters' transient, which disappears once they stabilize. The rest of the timeseries is almost identical.

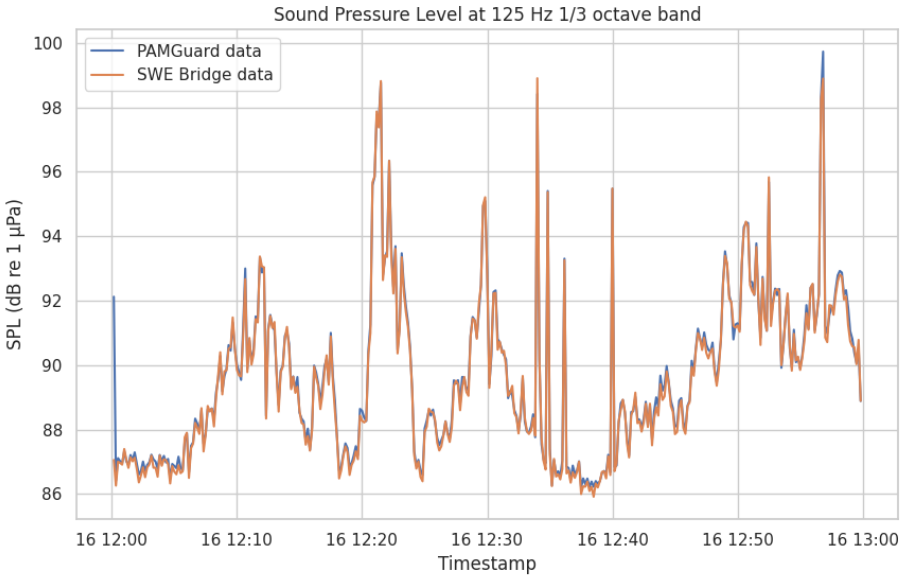


Figure 3.8: Timeseries produced by the SWE Bridge and PAMguard for the third-octave band centered at 125 Hz.

Results of both algorithms present a very high correlation, as shown in figure 3.9. The correlation coefficient between both timeseries ranges from 98.27 % to 99.29 %. Thus, it is possible to conclude that the implementation of the SPL algorithm within the SWE Bridge provides accurate results.

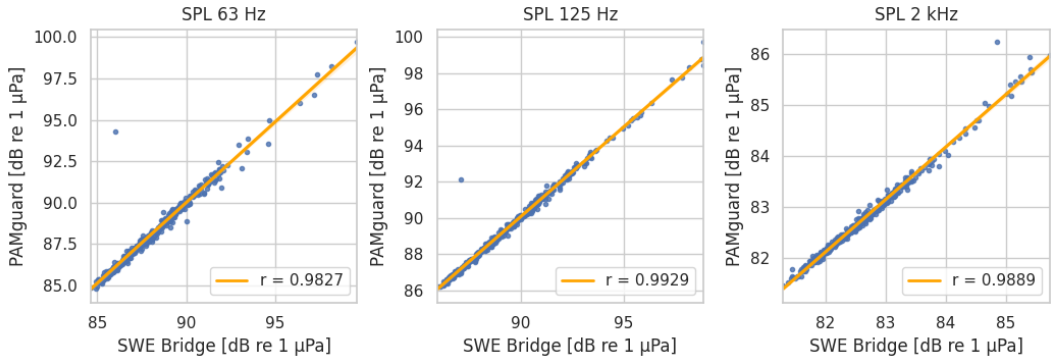


Figure 3.9: Correlation between SWE Bridge and PAMguard's SPL timeseries.

3.6 Conclusions

First, an overview on hydrophone acquisition chains is presented, enumerating the parameters that need to be taken into account when abstracting hydrophone interfaces to obtain the pressure signal.

Then, three different strategies to obtain ocean sound measurements at arbitrary frequency bands have been compared: Filter Bank, FFT and Goertzel's algorithm. The most optimal one, the FFT-based algorithm, has been selected and implemented within the SWE Bridge. Implementation considerations are discussed, such as averaging and windowing strategies (Welch and Bartlett's methods), handling of missing data and acoustic recordings.

Finally, the results of the algorithm has been validated by comparing its output with the specialized software PAMguard. The algorithm's outputs shows an excellent correlation (from 98.27 to 99.29 %). Thus, it is possible to conclude that the algorithm's implementation is accurate enough for ocean sound monitoring.

Chapter 4

Use Cases

The metadata model to abstract sensors, the interoperable architecture for sensor integration discussed in chapter 2 and the ocean sound algorithm discussed in chapter 3 has been deployed in various real-world scenarios using state-of-the-art observation platforms. Thus, in this chapter all the thesis objectives are evaluated in several real-world applications. The used platforms, listed in table 4.1, have heterogeneous operating systems, hardware resources, telemetry and communication interfaces.

Multiple sensors have been integrated using the SWE Bridge and SDFs/HSD to model their characteristics. The acquired data was archived and published using SWE-based services. The SWE Bridge could effectively abstract the underlying hardware and software characteristics, proving a common layer for interoperable sensor integration.

Platform Name	**Platform type**	**OS**	**Telemetry**	**Interfaces**
EGIM (CI)	Mooring	Linux	Ethernet	Serial, Ethernet
EGIM (Costof2)	Mooring	freeRTOS	n/a	Serial, Ethernet
OBSEA	Cabled Observatory	Linux	Ethernet	Serial, Ethernet
Wave Glider SV2	ASV	Linux	Iridium	Serial
Balizamar	Moored Buoy	Linux	Iridium	Serial
Sailbuoy	ASV	Windows	Iridium	Serial
SeaExplorer	Underwater glider	Linux	Iridium	Serial, Ethernet
INTMARSIS	Mooring	Linux	GSM	Serial
PECT	Moored Buoy	Linux	GSM	Serial
PIROS	DataLogger	freeRTOS	GSM	Serial, Analogue

Table 4.1: List of platforms where the SWE Bridge has been integrated

4.1 EGIM

The EMSO Generic Instrument Module (EGIM), depicted in figure 4.1, is a sensor system designed to provide long-term, accurate measurements in a wide range of

applications [75]. It has been designed bearing in mind the heterogeneity of the regional facilities that conform EMSO ERIC [76]. Thus, it can be deployed in moorings, at the seafloor, etc. It can be connected to other infrastructures such as cabled observatories (cabled mode) or it can be deployed autonomously (standalone mode) using its internal batteries.

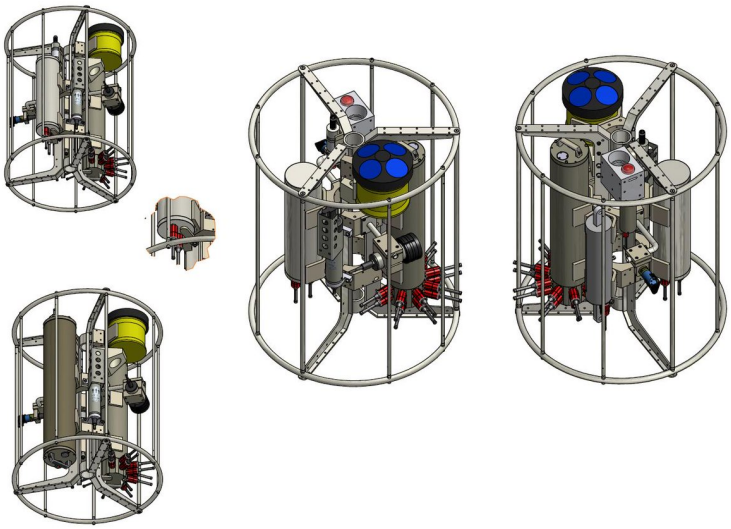


Figure 4.1: EMSO Generic Instrument Module (EGIM) drawings

The EGIM has an instrument package that gathers data covering scientific applications such as geoscience, physical oceanography, biogeochemistry and marine ecology. Currently, its instrument package is composed by the sensors enumerated in table 4.2:

Name	**Type**	**Manufacturer**	**Interface**	**Encoding**
Workhorse	ADCP	Teledyne RD	Serial	ASCII
SBE54	Tsunameter	Sea-Bird Electronics	Serial	XML
ECO NTU	Turbidimeter	WetLabs	Serial	ASCII
SBE37	CTD	Sea-Bird Electronics	Serial	ASCII
Aanderaa 4831	Oxygen Optode	Aanderaa Instruments	Serial	ASCII
icListen HF	Hydrophone	Ocean Sonics	Ethernet	WAV

Table 4.2: EGIM instrument package

EGIM has two modes of operation: cabled, and autonomous (depicted in figure 4.2). In autonomous mode the EGIM works on batteries with no external guidance, managing the equipment, acquiring and storing data internally. On the contrary, the cabled mode is the scenario where the EGIM is connected to a shore station through an underwater cable offering continuous power supply and communications. In this mode, the EGIM electronic core operates as a transparent junction box between the shore station and the instrument package, providing access to the sensors through

standard internet protocols (TCP/IP) and provides a serial to Ethernet converter for sensors with no Ethernet interface available. In cabled mode, the data acquisition is performed at the shore station by the EGIM's Cyber-infrastructure.

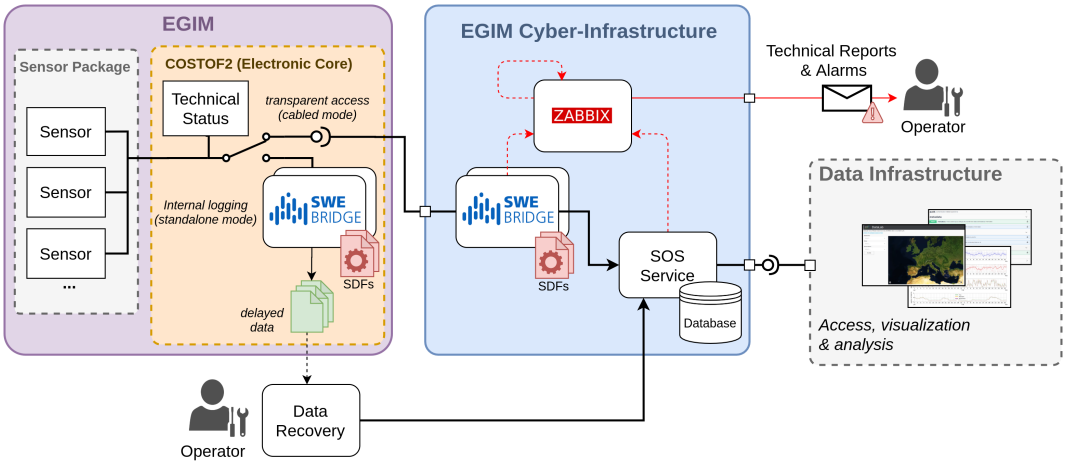


Figure 4.2: EGIM dataflow. The SWE Bridge was integrated both at the cyber-infrastructure and at the EGIM controller (COSTOF2). The switch in the EGIM represents the mode of operation selection: standalone or cabled. In both cases an instance of the SWE Bridge acquires sensor data. In cabled mode, the SWE Bridge also collects technical data and injects it to the Zabbix monitoring system.

4.1.1 EGIM Cyber-infrastructure

The EGIM CI (cyber-infrastructure) is a set of virtual machines containing all the components to acquire, archive, visualize and provide access to EGIM's data in real-time. It is an implementation of the interoperable architecture proposed in chapter 2. Each instrument is managed by an instance of the SWE Bridge, which acquires and injects data to the SOS service. The SOS service within the EGIM CI is used as a gateway to provide coherent access to EGIM's data and metadata to spatial data infrastructures through a standardized interface. It also contains a Zabbix monitoring service in order to provide real-time alarms and reports about the EGIM's status [77]. The paper A.4 explains in detail the integration of the EGIM's instrument package into the SWE framework. In the paper A.1, section 6.2 the details on the deployment are provided.

The EGIM CI has been successfully deployed during the EGIM deployment at OBSEA from December 2016 to April 2017. Additionally, the same CI has been replicated and successfully tested at PLOCAN (*Plataforma Oceánica de Canarias*) and at the Western Ionian EMSO facilities.

4.1.2 COSTOF2

The COSTOF2 is the electronic core of the EGIM, developed by Ifremer (*Institut français de recherche pour l'exploitation de la mer*). In standalone mode it is in charge of configuring, acquiring and archiving sensor data. It has a modular design based on the Silicon Lab's EFM32 Giant Gecko, a low-power, ARM Cortex M3 32-bit microcontroller. The COSTOF2 acquistion was historically performed using ad-hoc drivers. Due to its role as multi-sensor platform dozens of drivers have been coded for the COSTOF2.

Within the context of the ENVRIplus project, an exchange of personnel was carried out at Ifremer facilities at Brest (France) during April 2017. The purpose of this 4-week exchange of personnel was to improve COSTOF2's interoperability by integrating the SWE Bridge within its firmware.

Due to its intrinsic constrains (specially limited RAM and ROM), it was very challenging to integrate. There was less than 20 kBytes of RAM available to run multiple instances of the SWE Bridge. After several optimization efforts, the SWE Bridge was successfully integrated within the COSTOF2. The achieved average memory usage was below 2 kBytes, with a peak usage below 6 kBytes during the initialization (parsing of SensorML files).

A successful demonstration of a plug and play integration into the COSTOF2 of two different sensors was performed, during the workshop on Interoperability Technologies and Best Practices in Environment Monitoring Workshop, within the Sea Tech Week 2017 in Brest, France.

4.1.3 EGIM Acoustics

During the EGIM's deployment at PLOCAN test site, two months of acoustic recordings were acquired. This data has been processed offline in order to obtain meaningful underwater sound levels. The processing was performed using the SWE Bridge, taking advantage of its passive acoustics module, as detailed in A.2 section V-B.

Results

Deployments and tests performed with the EGIM sensor system provided an extremely valuable test bench to evaluate the proposed architecture. Due to the heterogeneous nature of sensors, protocols and interfaces used, the architecture's universality was tested in a very demanding real-world scenario.

The SDF ability to model and abstract sensor characteristics was demonstrated by interfacing from simple sensors (e.g. CTD, Oxygen Optode) to very complex

sensors (ADCPs). Moreover, new data encodings (such as XML) and specific cases not explicitly considered in the SensorML standard (e.g. start token) where encountered and properly addressed.

The SWE Bridge's ability to interface different formats and protocols (binary, ASCII, WAV, XML...) was also put to test.

Furthermore, the cross-platrofm nature of the SWE Bridge software was verified. The SWE Bridge was successfully compiled and deployed in two completely different environments: a Linux server and an ultra low-power microcontroller. The integration of the SWE Bridge into COSTOF2 proved challenging due to the few resources available in the system. After some optimization iterations, an extremely optimal resource usage was achieved.

Finally, the acoustic data acquired by EGIM was valuable to ensure that the architecture is not limited to real-time data processing, but it is also capable of processing acoustic recordings offline.

4.2 OBSEA

OBSEA, depicted in figure 4.3, is Cabled Observatory managed by the Universitat Politècnica de Catalunya [78], [20]. It is located 4 km off the coast of Vilanova i la Geltrú (Catalonia, Spain). It has a surface buoy and a seafloor node at 20 meters depth. It is a multi-parametric platform, acquiring scientific data from various sensors such as CTD, ADCP, hydrophone, seismometer, Weather Station and video camera. In order to address instrument heterogeneity and integration challenges, the SWE Bridge has been deployed in the OBSEA's acquisition infrastructure. It manages the dataflow of the hydrophone, the CTD and the Weather station (table 4.3). The OBSEA's data data flow is depicted in 4.4

Figure 4.3: OBSEA Cabled Observatory, seafloor node (left) and surface buoy (right).

Underwater noise levels in real time are provided by the SWE Bridge's embedded

acoustics module, as detailed in A.2 section V-A. Ocean sound data (SPL, SEL, and averaged RMS) are available at the OBSEA's SOS and ERDDAP services. Furthermore, OBSEA's underwater noise data is integrated in real-time within the EMODnet Physics data portal, using the SOS service as a gateway [79].

Name	Type	Manufacturer	Interface	Encoding
150WX	Weather Station	Airmar	Serial	ASCII, NMEA
SBE37	CTD	Sea-Bird	Serial	ASCII
NAXYS	Hydrophone	Bjørge	Ethernet	Binary

Table 4.3: OBSEA sensors managed by the SWE Bridge

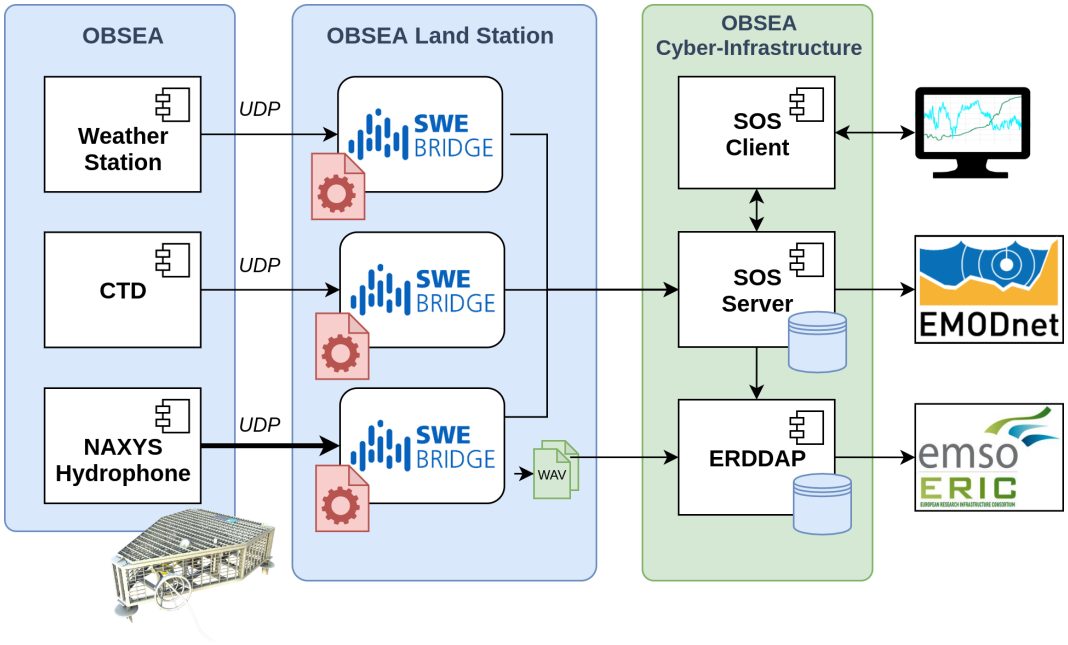


Figure 4.4: OBSEA data flow. Several instances of the SWE Bridge are deployed at the shore station, where they interface data streams coming from both surface and seafloor nodes. Data processed and acquired by the SWE Bridge is sent to OBSEA's SOS and ERDDAP services. Then, this data is harvested by EMSO ERIC and EMODNet using standard interfaces.

Results

The deployment of the architecture at OBSEA demonstrated its stability in continuous, long-term deployments. The SWE Bridge's ability to process hydrophone data in real-time was also demonstrated with this deployment. At a data management level, two different SDIs are directly coupled with the architecture using standard services (SOS and ERDDAP), without the need of custom-made connectors or harvesters. Thus, the proposed architecture has been demonstrated to be fully interoperable.

4.3 SeaExplorer

The SeaExplorer, depicted in 4.5, is a wingless underwater glider manufactured by the French company ALSEAMAR ALCEN. It is able to navigate autonomously for months, covering thousands of kilometres at depths up to 1000m [23]. It has a low-power Linux-based embedded controller with Iridium (satellite) communications. In the framework of the NeXOS project, the SWE Bridge was integrated within the SeaExplorer glider controller [80]. Two OGC PUCK-enabled sensors were integrated into the glider in order to demonstrate the interoperability of SEISI (Smart Electronic Interface for Sensor Interoperability) [81]. This interface uses the architecture proposed in chapter 2 to provide plug and play instrument integration.

A hydrocarbons sensor (Mini.1) and a smart passive acoustic hydrophone with built-in signal processing (A1) were integrated to the glider in a mission at Norwegian waters in June 2017 , as thoroughly detailed within the paper A.1 section 6.1. The architecture data flow is depicted in figure 4.6.

Both sensors implemented the PUCK protocol alongside their proprietary communication mechanism. Within the PUCK Payload, a Sensor Description File (SDF) was embedded, describing the instrument, its communication protocol and the mission. The SWE Bridge deployed within the SeaExplorer glider retrieved their respective SDF files from their PUCK Payloads and setup on-the-fly the acquisition process for the mission. Due to the intrinsic constraints of gliders, special attention was put on use of limited resources, specially the costly satellite communications to transmit data. Thus, the efficient EXI encoding was used to transmit a subsampled dataset in near real-time. An uncompressed full dataset was stored on-board for retrieval after the glider's recovery.

Figure 4.5: SeaExplorer glider (source www.alseamar-alcen.com)

Name	Type	Manufacturer	Interface	Encoding
A1	Hydrophone	NeXOS Consortium	Serial	ASCII, PUCK
Mini.1	Hydrocarbons	NeXOS Consortium	Serial	ASCII, PUCK

Table 4.4: Sensors integrated into the SeaExplorer Glider

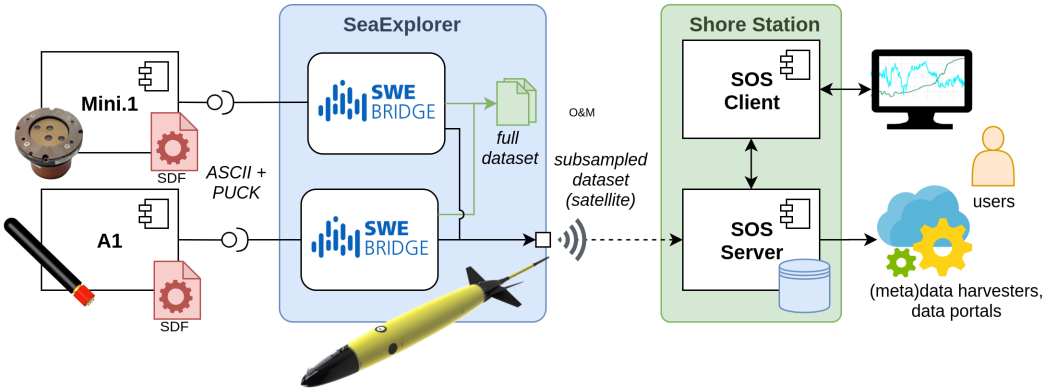


Figure 4.6: SeaExplorer workflow. Both Mini.1 and A1 hydrophones have their own SDF file embedded within their PUCK Payloads. The SWE Bridge downloaded the SDFs using the OGC PUCK Protocol and setup the acquisition. Data acquired by SWE Bridge, encoded in EXI format, was sent to the NeXOS SOS using satellite communications.

Results

Missions carried out with the SeaExplorer demonstrated the ability of the SWE Bridge to acquire data in real-world scenarios with resource-constrained observation platforms. The plug-and-play integration of PUCK-enabled sensors was successfully demonstrated. In this application the SWE Bridge's stability was crucial, since it was deployed on the vehicle's main controller. Any unexpected malfunction on the SWE Bridge could lead to a fatal software failure, causing the potential loss of the vehicle. Thus, the SWE Bridge robustness and stability were demonstrated.

Additionally, the subsampling feature was implemented and tested. This feature is vital for observation platforms with very limited telemetry that cannot send all data in real-time.

4.4 SailBuoy

SailBuoy, depicted in figure 4.7, is an Autonomous Surface Vehicle (ASV) manufactured by the Norwegian company Offshore Sensing AS [22]. It takes advantage of the wind to navigate, achieving long-term deployments of up to 6 months with minimal human intervention, delivering real-time data via Iridium satellite communications. Within the scope of the NeXOS project, the SWE Bridge was integrated into the SailBuoy's payload controller, a windows-based 64-bit Intel Atom (Bay Trail)

single-board computer.

The SWE Bridge interfaced the experimental Cbon sensor, a spectrophotometric seawater pH analysis and dissolved CO_2 detection (table 4.5). The aim of this mission was to demonstrate the feasibility of long term, in situ ocean acidification surveys based on ASV and spectrophotometric systems. The data flow is depicted in figure 4.8.

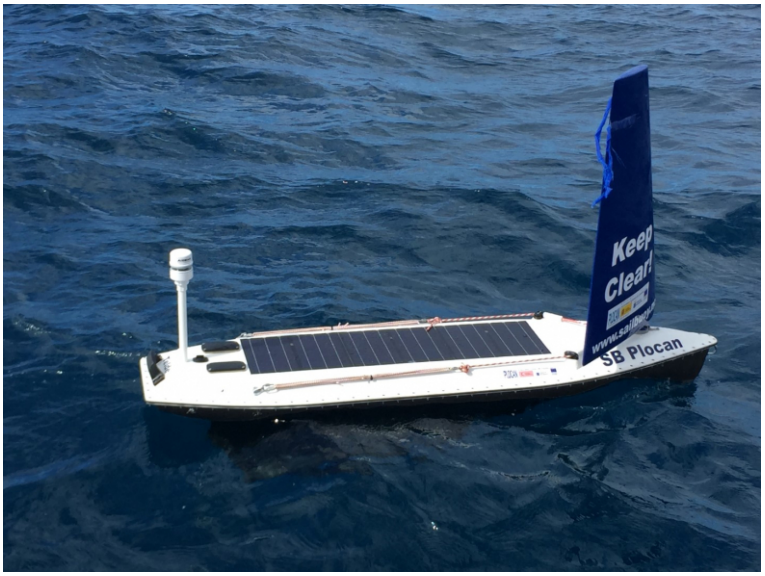


Figure 4.7: SailBuoy unmanned surface vehicle (source www.sailbuoy.no)

Name	Type	Manufacturer	Interface	Encoding
Cbon	pH and CO_2	NeXOS Consortium	Serial	ASCII

Table 4.5: Sensors integrated into the SailBuoy

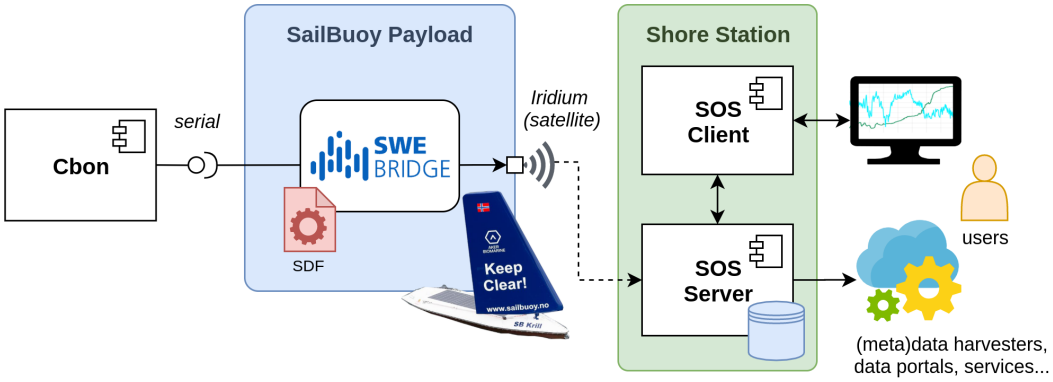


Figure 4.8: SailBuoy workflow. Cbon sensor is interfaced by the SWE Bridge, running on a Windows embedded computer. Since Cbon did not have PUCK capabilities, the SDF file was uploaded manually to the payload computer. Data acquired by the SWE Bridge is sent in EXI format via satellite communications to an SOS instance at the shore station.

Results

The main technical difficulty of the SailBuoy deployment was the need to port the SWE Bridge to a Windows environment. Using the Cygwin runtime environment it was successfully ported and its cross-platform nature successfully demonstrated.

4.5 SensorBox

Within the context of the NeXOS project, the SensorBox datalogger prototype was designed (figure 4.9). The SensorBox is an interoperable, low-power datalogger based on the SWE Bridge software. It is composed of a Linux single-board-computer (Raspberry Pi 3), a GPS module and an Iridium modem. It can control up to three serial sensors. Its versatility has been demonstrated in several deployments in both fixed and mobile platforms, such as the Balizamar buoy and Wave Glider ASV.

The main focus of the SensorBox is to obtain a SWE-compliant datalogger that is completely independent of the platform where it is deployed. This isolation may be a requisite for some observation platforms where the navigation systems are regarded as critical (e.g. a Wave Glider, see section 4.5.2). Although an increase of power consumption due to the redundant systems, more robustness and isolation is achieved in exchange.

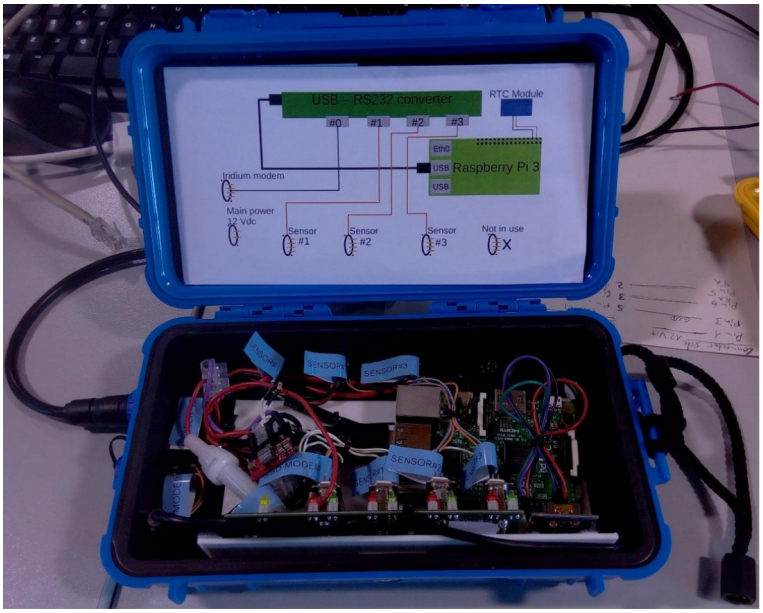


Figure 4.9: SensorBox datalogger

4.5.1 Balizamar Buoy

In the framework of the NeXOS project, the NeXOS A1 hydrophone was deployed at the Balizamar Buoy using the SensorBox datalogger off-the-coast of Gran Canaria, Spain. Sensor characteristics are listed in table 4.6. The Balizamar B1600S buoy is a moored surface autonomous ocean platform with open architecture able to integrate on-demand science-payloads according to needs in both coastal and open-ocean locations (figure 4.10). The central buoy hardware was modified in order to allocate all needed elements and components, with a waterproof hull along its main body as a container for battery pack, a solar regulator and a timer to supply power at the selected timing. Four solar panels where mounted in a 90 ° configuration on the main mast, in order to fully cover the 360 °. The waterproof datalogger (see section 4.5) was connected to the batteries and Iridium antenna (emulating open-ocean transmission) in the upper part of the buoy turret.

Figure 4.10: Balizamar buoy payload (left) and two Balizamar buoys equipped with a SensorBox.

The NeXOS A1 smart hydrophone was connecting via serial port [31] providing real-time underwater sound levels using the proposed architecture. In order to simulate open-ocean conditions, the buoy was equipped with Iridium communications. The acquisition was performed using a SensorBox datalogger. The data workflow was identical to the acquisition depicted at figure 4.12, but deployed at the buoy's payload instead of a Wave Glider (see next section for more details).

Name	Type	Manufacturer	Interface	Encoding
A1	Hydrophone	NeXOS Consortium	Serial	ASCII, PUCK

Table 4.6: Sensors into the Balizamar buoy

4.5.2 Wave Glider

The Wave Glider, depicted in figure 4.11, is an ASV manufactured by the US company Liquid Robotics. It is powered by wave and solar energy, delivering real-time data for up to a year with no fuel [21]. With the latest advancements in energy harvesting and propulsion, the Wave Glider is a persistent mobile data-gathering platform able to travel tens of thousands of kilometers, collect data in the most demanding conditions, and deliver this data in real-time.

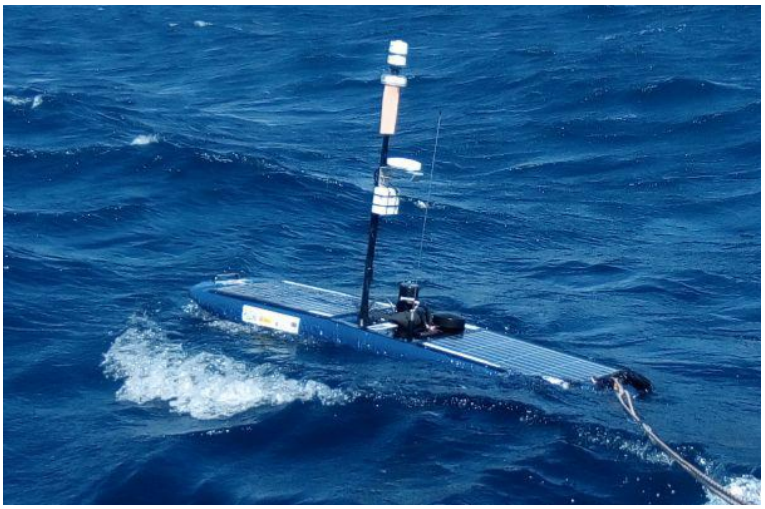


Figure 4.11: Wave Glider SV2 equipped with a SensorBox datalogger at the coast of Gran Canaria (Spain) during a mission in the context of the AtlantOS project.

Combining a Wave Glider SV2 and the SensorBox several missions where performed in the framework of the NeXOS, AtlantOS and MARCET projects [82, 83]. Through these missions sensors in table 4.7 where integrated using the proposed architecture.

Name	Type	Manufacturer	Interface	Encoding
NeXOS O1	CDOM, Phyocyanin, Chlorophyll-a	TriOS	Serial	ASCII, PUCK
NeXOS A1	Hydrophone	NeXOS	Serial	ASCII, PUCK
Cyclops 6K-T	Turbidity	Turner Designs	Serial	ASCII, PUCK
Cyclops 6K-RF	Refined Fuels	Turner Designs	Serial	ASCII
Cyclops 6K-Ca	Chlorophyll-a	Turner Designs	Serial	ASCII

Table 4.7: Sensors integrated into the Wave Glider using the SensorBox system

Within the NeXOS Project the A1 (smart hydrophone) and O1 (optical sensor) where integrated into a Wave Glider SV2. Both sensors are OGC PUCK-enabled, so their SDFs were embedded within their PUCK Payload, achieving plug and play sensor integration. Data acquired was sent in real-time to a shore station using Iridium satellite communications. In order to reduce the amount of data to be sent,

output data was encoded using EXI binary format.

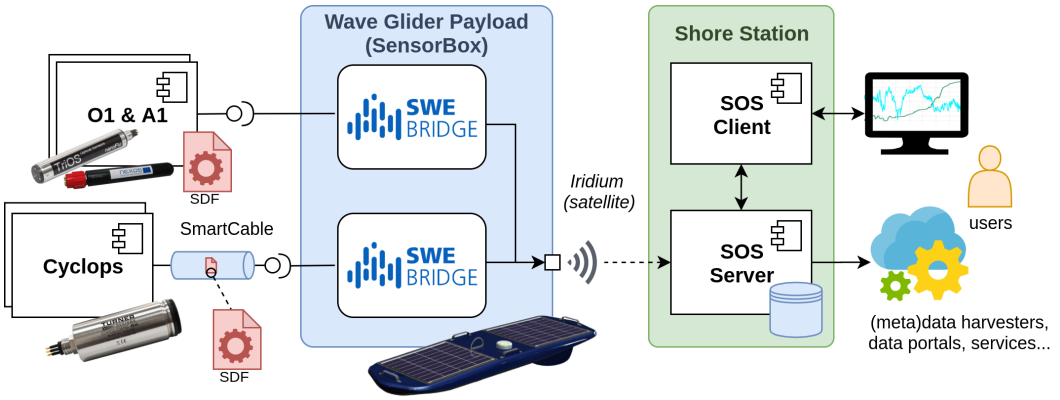


Figure 4.12: Wave Glider with a SensorBox workflow. A1 and O1 sensors had their SDFs embedded within their PUCK Payloads. The Cyclops 6k SDFs were stored in SmartCables. The SWE Bridge, deployed in the SensorBox, retrieved the SDFs via PUCK protocol and started the data acquisition. Data, encoded in EXI format, was sent to shore via satellite link.

The Cyclops 6k sensors (turbidity, chlorophyll and refined fuels) where commercial off-the-shelf sensors integrated to the Wave Glider within the scope of the AtlantOS project [83]. They were enhanced with three different SmartCables, where their SDFs were embedded to improve their interoperability (see section 2.3.4).

Results

Using the SensorBox datalogger, several missions were performed with the proposed architecture in both fixed and mobile platforms. The SmartCable's ability to enhance off-the-shelf sensors with PUCK capabilities was demonstrated using Cyclops 6k sensors. Furthermore, the SWE Bridge demonstrated its ability to interface different kinds of sensors and send data in real-time using the EXI format to save transmission costs.

4.6 INTMARSIS

The INTMARSIS project aims to monitor underwater seismic activity in real-time, allowing a precise estimation of actual earthquake scales. To achieve this goal a standalone seismic system with real-time telemetry was designed and tested [84, 85]. One further objective was to assess the usability of OGC compliant standardized acquisition chains in underwater seismic applications.

The INTMARSIS system, shown in figure 4.13, is composed of mainly two components, an ocean bottom seismometer (OBS) and a surface buoy. The communication between the OBS and the surface buoy is performed by the stainless steel mooring

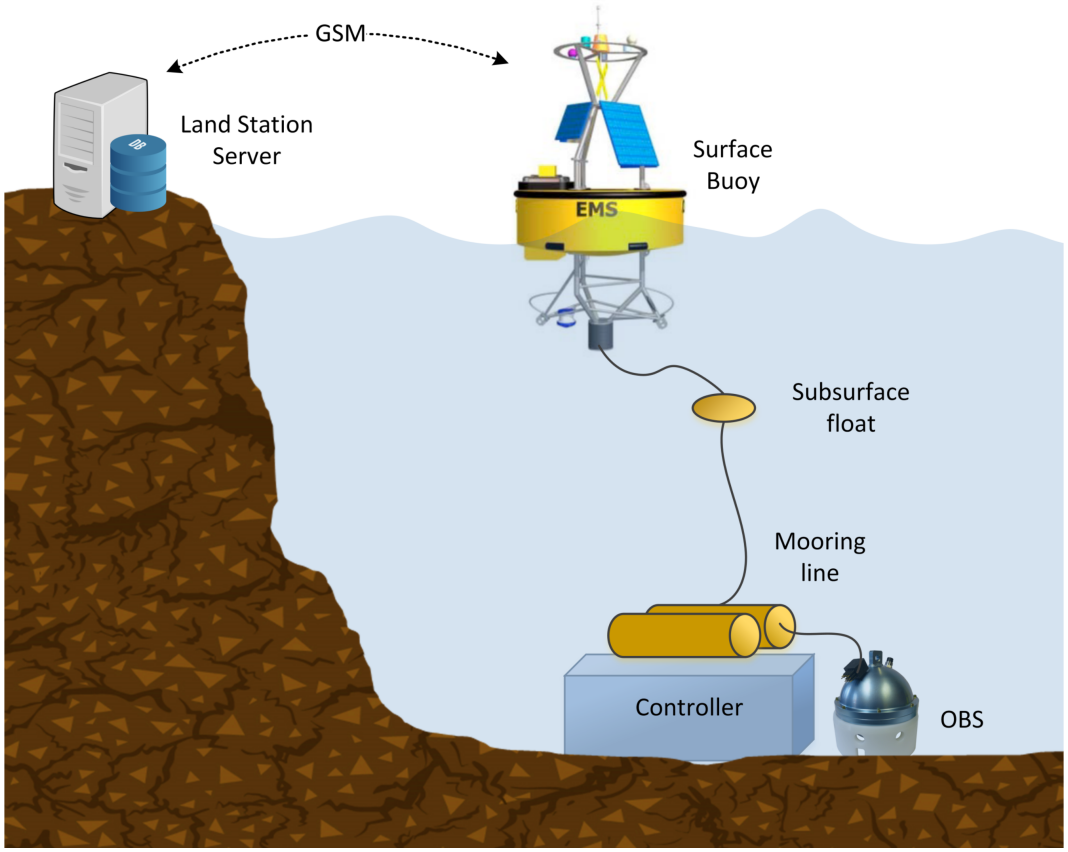


Figure 4.13: INTMARSIS System overview. At the seafloor the OBS (Ocean Bottom Seismometer) acquires seismic data, storing it locally. A subsampled set of data is sent through the mooring line using an inductive modem. The surface buoy receives the real-time subsampled seismic data, which is transmitted to the land station using a GSM link

line using inductive modems (SeaBird Electronics UIMM). The INTMARSIS system (depicted in 4.14), was deployed at OBSEA area, near the coast of Vilanova i la Geltrú (Barcelona, Spain).

The OBS acquires 3 channels (X, Y and Z axis), taking 125 samples per second (sps) with a precision of 24 bits. However, due to the low bandwidth provided by the inductive modems (1200 bps), the full data set is not transmitted in real-time, but stored locally. The OBS controller generates a subsampled data set at 25 sps which is transmitted thorough the inductive modem.

In figure 4.15 the INTMARSIS acquisition chain is depicted. The subsampled data set, alongside with OBS's technical data, is sent to the buoy through the inductive link. The inductive communication and the processing of the acquired data as well as internal sensors is performed by the master controller, hosted by the surface buoy. This software component is modelled as three different Virtual Instruments (VIs): OBS technical data, buoy technical data and a peak detector. The SWE Bridge

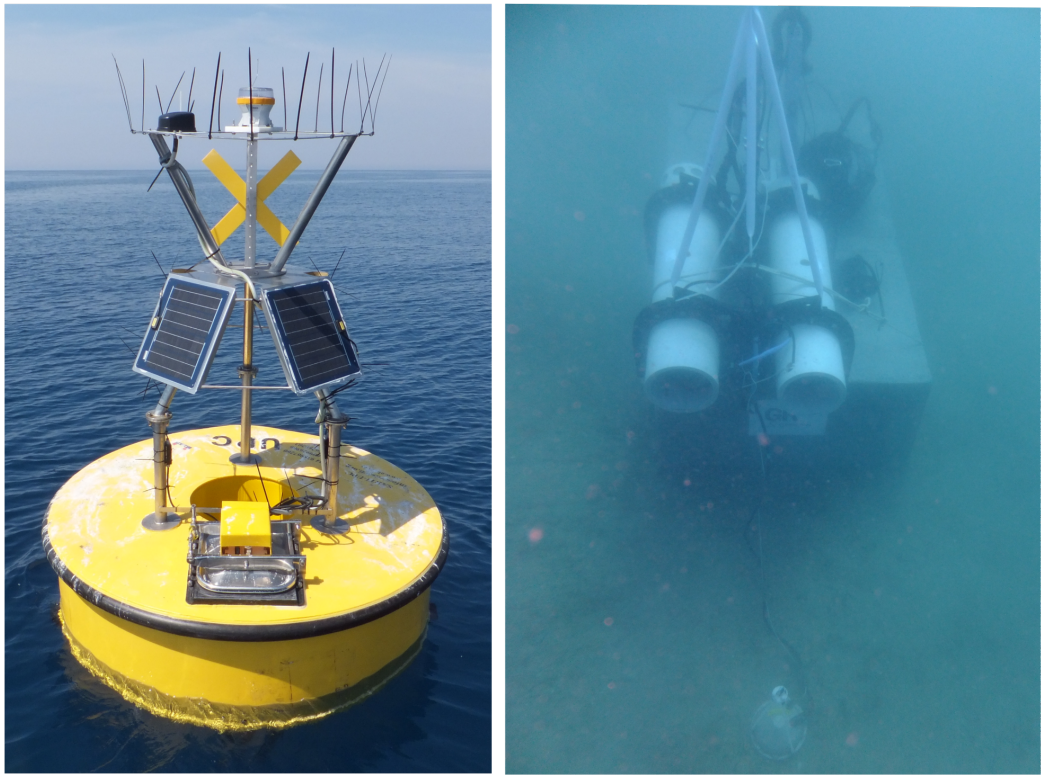


Figure 4.14: INTMARSIS buoy (left) and INTMARSIS Ocean Bottom Seismometer (right).

communicates to these VIs, managing and acquiring their data. More details are provided in the paper A.1., section 6.3.

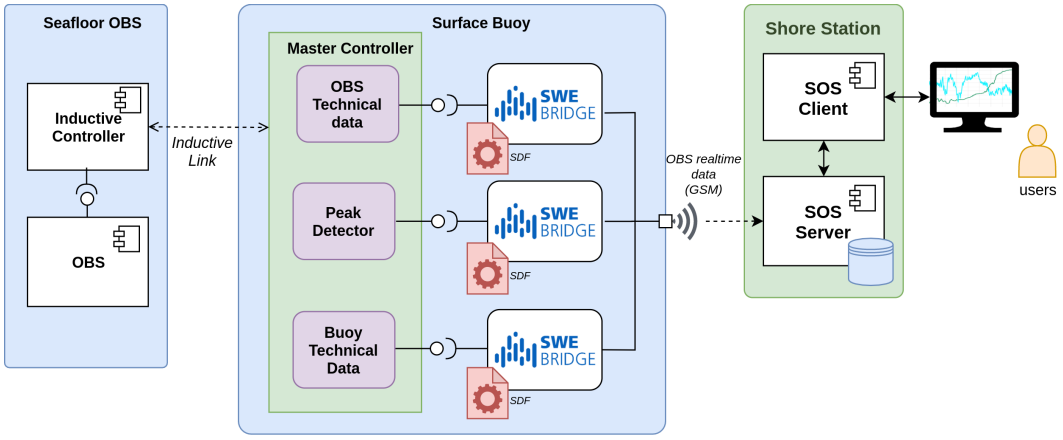


Figure 4.15: INTMARSIS workflow. Data from the OBS is transmitted through an inductive link to the surface buoy. The master controller software gathers the data from the OBS and technical data to generate three data streams. The SWE Bridge interfaces these Virtual instruments and send their data to an SOS service.

Results

The main outcome from the INTMARSIS project was the architecture's ability to integrate not only physical, but also virtual instruments. As long as it has a communications interface, an SDF can abstract its characteristics and SWE Bridge can interface it, acquiring its data.

4.7 PECT

Estuaries are coastal indentations where freshwater mixes with seawater, each one with its unique hydrology and physical exchange characteristics. Given projected changes in river flow to coastal regions due to climate change and increasing human freshwater demands, it is necessary to determine the role that hydrology plays in regulating the biochemistry of estuaries.

Figure 4.16: Besòs Integrated Biochemistry Observatory layout (top), coastal buoy (bottom left) and river station (bottom right)

The PECT project (*Projecte d'Especialització i Competitivitat Territorial, Litoral Besòs Territori Sostenible*) aims to achieve a sustainable usage of hydrological resources in the Besòs estuary (Barcelona, Spain). It funded by the European Union and involves local authorities along with research institutions.

Within the PECT project, a new integrated biochemistry observatory has been deployed in the Besòs estuary: The Besòs Integrated Biochemistry Observatory (figure 4.16). It is composed of two nodes: a coastal buoy and a river station. Both nodes are equipped with a multi-sensor system to monitor water quality and the interactions between the hydrology and the estuary's biochemistry.

The coastal buoy is located a few hundred meters away from the river's mouth. Its objective is to monitor the interaction patterns and exchange characteristics of the Besòs river. The river station is located few hundred meters upriver. It consists on a manhole and a pipe, connecting the manhole to the river. Using an electric pump water from the river is driven to the manhole, where the water is analized.

Name	Type	Manufacturer	Interface	Encoding
SA8606.101	Water Quality Probe	B&C Electronics	Serial	ASCII
Cyclops C3*	Fluorometer	Turner Designs	Serial	ASCII
150WX*	Weather Station	Airmar	Serial	ASCII, NMEA

Table 4.8: Sensors integrated into both Besòs Integrated Biochemistry Observatory nodes. Weather station and fluorometer are installed only in the coastal buoy

Both nodes have a dedicated data acquisition system based on an embedded linux computer (raspberry pi), where several instances of the SWE Bridge are deployed. Each SWE Bridge instance controls the power cycle and acquisition of a sensor. Data acquired by the SWE Bridge is stored in O&M format and sent in real-time to a dedicated SOS Service. Alongside scientific data, technical information is sent in real-time to a Zabbix monitoring service.

The telemetry to shore station uses a SierraLink modem, providing GSM communications. Using the SWE Bridge's power management modules, the power cycle of the sensors is managed, in order to save power and improve the system's autonomy. Moreover, using appropriate SDF file not only the sensors are managed, but also the electrical pump which draws water from the river to the manhole in the river station.

Results

Within the PECT project the SWE Bridge ability to manage sensor power cycle was successfully tested. Due to the underground location of the river station, GSM coverage was not guaranteed. Nevertheless, the proposed architecture could autonomously manage despite the intermittent communication link.

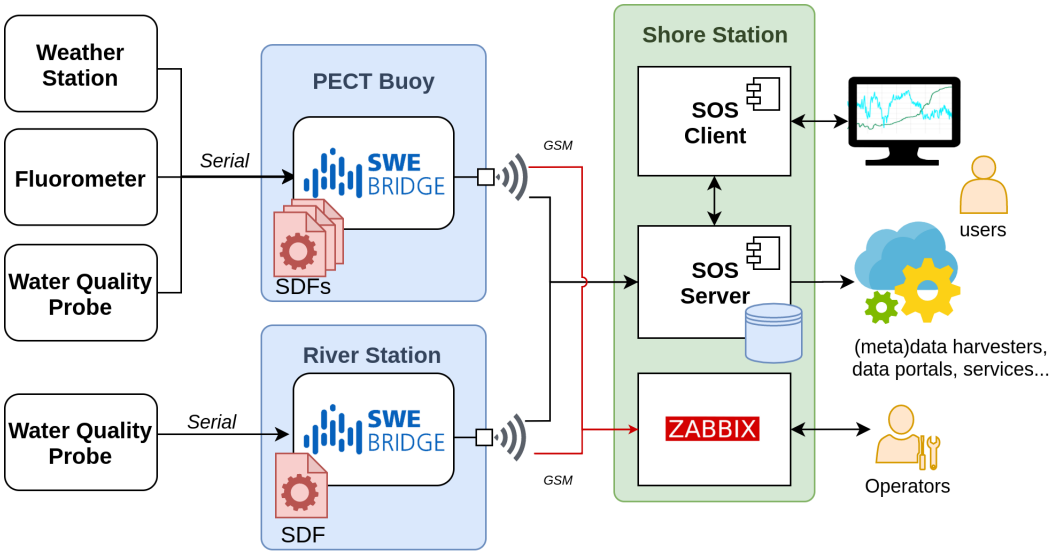


Figure 4.17: PECT data workflow. The SWE Bridge is deployed at the buoy and river station, managing their respective sensors. Alongside scientific data, technical information is sent in real-time to a zabbix service, where the operators monitor the system's operation.

4.8 PIROS

The PIROS (*Plataforma Integrada Remota para la Observación de Sensores*) is a commercial datalogger designed by the SensorLab company, based at Las Palmas de Gran Canaria, Spain. The system aims to be an interoperable, ultra-low power datalogger for scientific sensors. The system is based on the Texas Instrument's TM4C1294NCPDT microcontroller (see table 4.9). The PIROS system has a modular design based on extension cards that can be configured to adapt the system to any deployment (figure 4.18). Each extension card provides additional interfaces and/or telemetry options. Using these cards the system can interface and control up to dozens of sensors.

The SWE Bridge has been integrated within PIROS as the acquisition core, using SDF descriptions for each sensor. Moreover, the analogue acquisition, the sensor's power cycle and the internal sensors are also managed by the SWE Bridge. Using an SDF for each sensor different acquisition patterns can be configured, such as deep sleep, acquisition bursts as well as instrument power cycle.

An API was also developed to give to the PIROS core full control over the SWE Bridge. As a part of this API, an asynchronous notifications system was created, providing detailed information of the SWE Bridge status to the PIROS core in real-time.

The PIROS system has been used in several commercial deployments, interfacing all kinds of heterogeneous sensors. Some of these sensors are shown in table 4.10

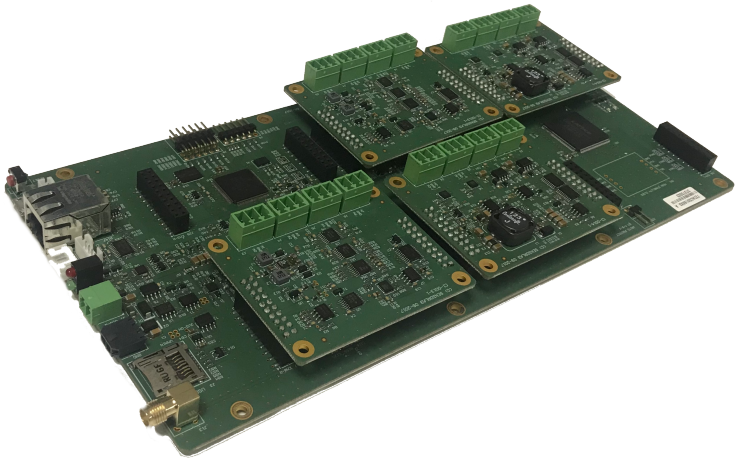


Figure 4.18: PIROS system with its extension cards

PIROS Specifications	
Architecture	Cortex M4
RAM	256 kBytes
CPU	120 MHz
Operating System	freeRTOS

Table 4.9: PIROS System Specifications

Results

The integration in the PIROS system proves that the software is robust and flexible enough to be used in industrial-grade systems. SensorLab has been using the PIROS platform in many applications, proving a reliable tool to interface heterogeneous sensors. Several new features were designed to push the SWE Bridge to an industrial-grade quality, such as ability to interface analog sensors (ADC), acquisition in burst mode, power management to control the sensors power cycle and deep sleep options among others.

The SWE Bridge core was also greatly improved in the process, making a single instance to control several sensors, providing an API for external control and including a notification system.

Name	Type	Manufacturer	Interface	Encoding
SAMI2 CO	pCO_2	Sunburst Sensors	Serial	ASCII
SAMI2 pH	pH	Sunburst Sensors	Serial	ASCII
SBE 37-SIP	CTD	Sea-Bird Electronics	Serial	ASCII
Aanderaa 4835	Oxygen Optode	Aanderaa	Serial	ASCII
GMX501	Weather Station	MaxiMet	Serial	ASCII, NMEA
Cyclops-7F	Turbidity	Turner Designs	Analogue	-
CO_2 Pro CV	CO_2	Pro-Oceanus	Serial	ASCII
Battelle pCO_2	pCO_2	Seaology	Serial	ASCII

Table 4.10: Sensors integrated using the PIROS System

4.9 Conclusions

Through this chapter, the deployment of the interopeable architecture into 10 observation platforms has been discussed. These platform have very heterogeneous capabilities, operating systems and resources. Nevertheless, the SWE Bridge could be deployed in all of them, demonstrating its flexibility and cross-platform nature. Morevoer, more than 30 sensors have been abstracted and interfaced, ranging from commercial-of-the-shelf sensors, to experimental ones. Thus, it is possible to conclude that the proposed architecture has the ability to integrate sensors into spatial data infrastructures, regardless of the protocols, observation platforms, operating systems and communications link.

Chapter 5

Conclusions and Future Work

5.1 Sensor Abstraction Mechanism

This work largely relies on the SensorML standard to model and abstract sensors, with emphasis on hydrophones. This standard provides a good framework to generate sensor descriptions in a robust and machine-understandable manner. Identifiers, characteristics, sensor commands, interfaces and more can be effectively described and encoded using this standard. A very demanding application is hydrophone abstraction due to the high number of characteristics required (sensitivity, ADC number of bits, data stream, etc.). Even in this demanding application, the SensorML standard provided a framework to effectively abstract them.

SensorML's modular approach grants the ability to describe complex acquisition workflows. A set of components can be easily declared, configured and connected, generating user-defined workflows. The standard also grants inheritance, referencing and configuration mechanisms required to generate such workflows.

The standard is very open, providing a tool to abstract almost every sensor and any sensor-related operation. However, this openness proves a double-edged sword, since the same feature can be encoded in different ways.

One of the most complicated aspects to unambiguously define is sensor's data streams, which are encoded using the SWE Common Data Model standard within SensorML documents. Some minor but important aspects are missing. For instance, the standard does not contemplate a way to define a stream that starts with a fixed token (start token) or to model a data stream that combines variable elements (data) with fixed fields (tokens, units, etc.) that would simplify their decoding and parsing.

Another weakness of the SensorML standard is its verbosity. Usually lots of subelements are required to encode simple features, adding a lot of redundancy with little or no benefit. In example, there are multiple mandatory intermediate elements

that do not provide extra information. Another verbosity issue is that names, labels, identifiers and definition are sometimes superfluous and may even have contradictory information. This verbosity also increases the complexity to reference elements within the same document.

To overcome the limitations of the SensorML standard, within this thesis the Sensor Deployment File (SDF) and Hydrophone SensorML Descriptions (HSD) data models have been proposed. These metadata models further restrict SensorML description and organize metadata in a coherent manner. Without these restricted data models, it would be nearly impossible to achieve syntactic and/or semantic interoperability in SensorML-based machine-to-machine applications.

Using SDF and HSD it is possible to generate SensorML-compliant documents that unambiguously define sensor characteristics that otherwise would be too open and ambiguous to be interpreted. Furthermore, it also addresses missing aspects in the standard, such as a way to encode start tokens or data streams with fixed fields. Thus, these metadata models are an effective tool to model and abstract sensor characteristics, fulfilling the objective 1 of this thesis.

These descriptions also contemplate the generation and configuration of complex workflow to acquire and process sensor data. Thus, the data acquisition and processing are effectively configured and managed using the deployment's metadata, achieving a metadata-driven acquisition. A metadata-driven acquisition provides key features for FAIR data management, such as traceability, reproducibility and re-usability.

5.2 Sensor Integration Architecture

Within this thesis, on-the-fly sensor integration is achieved by combining various standards, protocols into an interoperable architecture. Using this architecture a wide variety of oceanographic sensors have been deployed in observation platforms and their data integrated into scientific data infrastructures.

One of the key components of the architecture is the SWE Bridge universal driver, which interprets SDF and HSD files to interface sensors. Its ability to abstract sensor resources has been demonstrated with more than 30 scientific sensors. These sensors have a wide variety of interfaces and communications protocols, ranging from analog sensors up to broadband hydrophones (see chapter 4). The SWE Bridge combines SDF/HSD descriptions with the OGC PUCK Protocol to achieve plug and play sensor integration, fulfilling the objective 2 of this thesis.

On the other hand, the SWE Bridge's flexibility and universality have been demonstrated in its deployment in more than 10 observation platforms, including moored buoys, cabled observatories, autonomous surface vehicles, underwater gliders,

low-power dataloggers and more. Due to its design based on hardware abstraction, it can run seamlessly in platforms using Linux, Windows and even in microcontroller-based systems. Its efficiency and optimized design has been validated in platforms with very limited resources such as the COSTOF2 and the PIROS commercial datalogger.

During all the deployments and missions discussed on chapter 4, the SWE Bridge's functionalities have been continuously improving and adapting to different deployment strategies. Currently, it is a flexible and robust tool for sensor integration, in situ processing and data ingestion.

5.3 Real-time Ocean Sound Processing

The proposed interoperable architecture is also suitable for ocean sound monitoring. The SWE Bridge has the ability to interface almost any hydrophone based on standard metadata definitions (HSD). Furthermore, it has a built-in embedded ocean sound module to process data in real-time. Thus, it can be used as an out-of-the-box universal tool for in situ ocean sound monitoring, regardless of the underlying platform and hydrophone characteristics. This standardized approach significantly reduces the time and costs associated to a hydrophone deployment and data management. Thus, the algorithm fulfills the objective 3 of this thesis: an optimized algorithm for in situ ocean sound measurements from resource-constrained platforms.

In addition to its embedded processing capabilities, it is also able to provide acoustic recordings for further off-line processing. In order to overcome the intrinsic limitations of WAV format, it uses the ID3 informal standard to embed metadata into WAV files. Thus, contextual information is added to the acoustic dataset, ensuring data re-usability and traceability.

5.4 Future Work

Future work on sensor abstraction based on SensorML could be to extend the standard and create normative schemas instead of SDF and HSD soft-typed templates. Another future line of work in SensorML could be to generate software tools to ease their generation, hiding the complexity of the SensorML standard to the user. Such a tool would potentially use the SWE Bridge Model to understand its functionalities.

The use cases presented within this thesis focus on deployments or missions with one observation platform hosting a few sensors. Although the architecture have largely demonstrated its benefits, the generation and delivery of SDFs/HDFs are currently manually performed. In large-scale scenarios (tens of platforms, hundreds of sensors) the manual generation and delivery of metadata files is not practical.

Thus, a future line of work could be to generate software solutions to automate the generation, versioning and delivering of SDF/HSD files.

Another line of research is to explore other data sharing standards and services. Currently, the architecture works mainly with Sensor Observation Service and ERDDAP. However, new technologies for data sharing are emerging constantly. To boost the SWE Bridge's interoperability with new data services, more data delivery systems could be included, such as a SensorThings API connector. Another future line of work is to directly generate datasets with their associated metadata (e.g. NetCDF files). This approach could automate the integration of scientific data into existing and future data catalogues.

Since a hydrophone abstraction layer and real-time reception of data streams is already implemented, future work on acoustics could focus on the integration of additional algorithms within the SWE Bridge. New passive acoustics algorithms could expand its functionalities beyond its current capabilities, such as cetacean detection and sound source localization. Including these new algorithms as modules should be straight-forward, since the framework for configuring and abstracting hydrophones is already implemented.

Publications associated to the thesis

Journals

(IEEE ACCESS '21) Martínez, E.; A. García-Benadí; Delory, E. ; Toma, D.M.; Gomáriz, S. ; Del Rio, J. *Metadata-Driven Universal Real-Time Ocean Sound Measurement Architecture* IEEE Access. 2021. Volume: 9. Number: 2021. pp.: 28282 - 28301.

(SENSORS' 17) Martínez, E.; Toma, D.M.; Jirka, S.; Del Rio, J. *Middleware for plug and play integration of heterogeneous sensor resources into the sensor web.* Sensors. 2017. Volume: 17. Number: 12.

(IEEE ACCESS '20) Del Rio, J.; Nogueras, M.; Toma, D.M.; **Martínez, E.**; Artero, C.; Bghiel, I.; Cadena-Muñoz, F.J.; Garcia-Benadí, A.; Sarria, D.; Aguzzi, J.; Masmitja, I.; Carandell, M.; Olive, J.; Gomariz, S.; Santamaria, J.; Manuel, A. *Obsea: a decadal balance for a cabled observatory deployment.* IEEE Access. 2020. Volume: 8. Number: 2020. pp.: 33163 - 33177.

(JOE'17) Del Rio, J.; Toma, D.M.; **Martínez, E.**; O'Reilly, T.C.; Delory, E.; Pearlman, J.; Waldmann, C.; Jirka, S. *A sensor web architecture for integrating smart oceanographic sensors into the semantic sensor web.* IEEE journal of oceanic engineering. 2018. Volume: 43. Number: 4. pp.: 830 - 842.

(MEASUREMENT '18) Toma, D.M.; Masmitja, I.; Del Rio, J.; **Martínez, E.**; Artero, C.; Casale, A.; Figoli, A.; Pinzani, D.; Cervantes, P.; Ruiz, P.; Memè, S.; Delory, E. *Smart embedded passive acoustic devices for real-time hydroacoustic surveys.* Measurement. 2018. Volume: 125. pp.: 592 - 605.

Conferences

(IMDIS'18) Martínez, E.; Toma, D.M.; Del Rio, J. Sensor metadata for automated integration of sensor resources into research data infrastructures. Bolletino di geofisica teorica ed applicata: An International Journal of Earth Sciences: IMDIS 2018 International Conference on Marine Data and Information Systems 5-7 November, 2018, Barcelona (Spain), vol. 59, supplement 1. 2018. pp.: 194 - 195.

(EGU'18) Martínez, E.; Garcia-Benadí, A.; Toma, D.M.; Del Rio, J. Effortless Integration of Underwater Noise Measurements into EMODnet data portal through Sensor Web Standards. Geophysical research abstracts, vol. 20, 2018. 2018. pp.: 13103 - 13103.

(OCEANS'17) Martínez, E.; Toma, D.M.; Del Rio, J.; Jirka, S. Remote configuration service for marine observation platforms through Sensor Web Enablement components. OCEANS 2017 - Aberdeen: 19-22 June 2017. Institute of Electrical and Electronics Engineers (IEEE). 2017.

(SENSOR WEB'17) Martínez, E.; Toma, D.M.; Del Rio, J.; Jirka, S. SensorML based Plug & Play Sensor Integration into Research Data Infrastructures. Geospatial Sensor Webs 2017. Münster, Germany, August 28 - 30, 2017.

(SAS'16) Martínez, E.; Toma, D.M.; Trullols, E.; Del Rio, J.; Pinzani, D.; Delory, E.; Jirka, S.; Pearlman, J. NeXOS A1: Smart hydrophone integration into the sensor web enablement framework. SAS 2016: IEEE Sensors Applications Symposium: Catania, Italy: April 20-22, 2016: proceedings book. 2016. pp.: 1 - 3.

(AGU'19) Delory, E.; **Martínez, E.**; Toma, D.M.; Del Rio, J.; Caudet, E.; Waldmann, C.; Barrera, C.; Dañobeitia, J.; Llinás, O. Towards web-enabled ocean sensor networks. AGU Fall Meeting 2019 Abstracts.

(IMDIS'18) Toma, D.M.; **Martínez, E.**; Del Rio, J.; García, Ó.; Dañobeitia, J. Interoperable data management and instrument control experiences with the EMSO generic instrument module at OBSEA. Bolletino di geofisica teorica ed applicata: An International Journal of Earth Sciences: IMDIS 2018 International Conference on Marine Data and Information Systems 5-7 November, 2018, Barcelona (Spain), vol. 59, supplement 1. 2018. pp.: 155 - 157.

(OCEANS'17) Toma, D.M.; **Martínez, E.**; Del Rio, J.; Bghiel, I.; García, Ó.; Dañobeita, J.; Lantéri, N. Seamless integration of EMSO generic instrument

module into the internet using sensor web components based on OGC SWE framework. OCEANS 2017 - Aberdeen: 19-22 June 2017. Institute of Electrical and Electronics Engineers (IEEE).

(IMKEO'17) Toma, D.M.; Del Rio, J.; **Martínez, E.**; Casale, A.; Figoli, A.; Pinzani, D.; Cervantes, P.; Ruiz, P.; Delory, E.; Memè, S. An embedded passive acoustic device for realtime hydroacoustic surveys. Proceedings of the IMEKO TC4 Symposium 2017, Iasi, ROMANIA, 2017. 2017.

(IMEKO'17) Toma, D.M.; Del Rio, J.; **Martínez, E.**; Delory, E.; Pearlman, J.; Waldmann, C.; Jirka, S. Interoperable data management and instrument control architecture for ocean observing systems. 21st IMEKO TC4 International Symposium and 19th International Workshop on ADC Modelling and Testing Understanding the World through Electrical and Electronic Measurement. 2016. pp.: 262 - 266.

(AGU'19) Rodero, I.; Bardají, R.; Piera, J.; **Martínez, E.**; Favali, P.; Dañobeitia, J. A flexible cyberinfrastructure for integrating data from large-scale ocean observing systems and delivering online added-value services. AGU Fall Meeting 2019 Abstracts.

(AGU'19) Del Rio, J.; Nogueras, M.; Toma, D.M.; **Martínez, E.**; Masmitja, I.; Carandell, M.; Gomariz, S.; Olive, J. Obsea, ten years of coastal ocean monitoring and test site observatory. AGU 100 Fall Meeting 2019.

(EGU'17) Jirka, S.; Del Rio, J.; Toma, D.M.; **Martínez, E.**; Delory, E.; Pearlman, J.; Rieke, M.; Stasch, C. SWE-based observation data delivery from the instrument to the user - sensor web technology in the NeXOS project. Geophysical research abstracts, volume 19: EGU General Assembly 2017. European Geosciences Union (EGU). 2017.

(EGU'17) García, Ó.; Toma, D.M.; Dañobeita, J.; Del Rio, J.; Bartolome, R.; **Martínez, E.**; Nogueras, M.; Bghiel, I.; Lanteri, N.; Francois Rolin, J.; Beranzoli, L.; Favali, P. Acquisition system for the "EMSO Generic Instrument Module" (EGIM) and analysis of the first data obtained during its deployement at OBSEA (Spain). Geophysical research abstracts, volume 19: EGU General Assembly 2017. European Geosciences Union (EGU). 2017.

(ASLO'17) Del Rio, J.; Jirka, S.; Toma, D.M.; **Martínez, E.**; Pearlman, J.; O'Reilly, T.C. Nexos contributions to end-to-end data flow in marine sensor systems. Mountains to the sea : ASLO Aquatic Sciences Meeting 2017 M2C: Meeting abstracts. 2017.

(SENSOR WEB'16) Toma, D.M.; Del Rio, J.; **Martínez, E.**; Delory, E.; Jirka, S.; Pearlman, J.; Waldmann, C. Applying OGC sensor web enablement to ocean observing systems. Geospatial Sensor Webs 2016. Proceedings of the Geospatial Sensor Webs Conference 2016 : Münster, Germany, August 29 - 31, 2016. 2016. pp.: 1 - 5.

(SENSOR WEB'16) Toma, D.M.; Del Rio, J.; Cadena-Muñoz, F.J.; Bghiel, I.; **Martínez, E.**; Nogueras, M.; García, Ó.; Dañobeita, J.; Sorribas, J.; Casas, R.; Piera, J.; Bartolomé, R.; Bardají, R. OGC SWE-based Data Acquisition System Development for EGIM on EMSODEV EU Project. Geospatial Sensor Webs 2016. Proceedings of the Geospatial Sensor Webs Conference 2016 : Münster, Germany, August 29 - 31, 2016. 2016. pp.: 1 - 5.

(AGU'16) Delory, E.; Del Rio, J.; Jirka, S.; Toma, D.M.; **Martínez, E.** Efficient sensor integration on platforms (NeXOS). 2016 AGU Fall Meeting. American Geophysical Union (AGU).

(AGU'16) Pearlman, J.; Castro, A.; Corradino, L.; Del Rio, J.; Delory, E.; Garello, R.; Heuermann, R.; **Martínez, E.**; Pearlman, F.; Rolin, J.; Toma, D.M.; Waldmann, C.; Zielinski, O. Multifunctional web enabled ocean sensor systems for the monitoring of a changing ocean. Geophysical Research Abstracts. 2016.

(AGU'16) Dañobeita, J.; García, Ó.; Bartolomé, R.; Del Rio, J.; Cadena-Muñoz, F.J.; Toma, D.M.; Bghiel, I.; **Martínez, E.** European Multidisciplinary seafloor and the Observatory of the water column for Development. The setup of an interoperable generic sensor module. 2016 AGU Fall Meeting. American Geophysical Union (AGU). 2016.

Presentations at Workshops

(OI'18) Martínez, E.; Toma, D.M.; Del Río, J. OGC PUCK for sensor Interoperability. Interoperability Technologies for sharing ocean instruments and real-time data in the framework of Oceanology International 2018. 15th March, London.

(MARTECH'18) Martínez, E.; Toma, D.M.; Del Río, J. Sensor web enablement implementations in marine observation platforms. Instrumentation Viewpoint 2018, núm. 20. 2018. pp.: 48 - 48.

(SEATECHWEEK'18) Martínez, E.; Toma, D.M.; Del Rio, J.; O'Reilly, T.C. Sensor Web Enablement Implementations in Marine Observation Platforms.

Interoperability Technologies and Best Practices in Environment Monitoring Workshop in the framework of the Sea Tech Week, 10-11th October 2018, Brest.

(MARTECH'16) **Martínez, E.**; Toma, D.M.; Del Rio, J.; García, Ó. SWE bridge: software interface for plug & work instrument integration into marine observation platforms. MARTECH 2016: 7th International Workshop on Marine Technology, Barcelona October 26th, 27th and 28th, 2016. pp.: 123 - 124.

(SEATECHWEEK'16) Toma, D.M.; **Martínez, E.**; Del Rio, J.; O'Reilly, T.C. Standard-based middleware for marine sensor networks. A connected ocean : The challenge of observation data integration. 11–13th Oct 2016 - Brest, France : an international conference in the framework of the 10th edition of Sea Tech Week : Book of abstracts. 2016.

(SEATECHWEEK'16) Dañobeita, J.; García, Ó.; Bartolomé, R.; Del Rio, J.; Cadena-Muñoz, F.J.; Toma, D.M.; Bghiel, I.; **Martínez, E.** European Multidisciplinary seafloor and the Observatory of the water column for Development. The setup of an interoperable generic sensor module. 2016 AGU Fall Meeting. American Geophysical Union (AGU). 2016.

(MARTECH'18) Del Rio, J.; Andre, M.; Folegot, T.; Van Der Schaar, M.; Gorringe, P.; Novellino, A.; Garcia-Benadí, A.; **Martínez, E**. Dataflow of underwater noise measurements: from OBSEA to EMODnet. Instrumentation Viewpoint 2018, núm. 20. 2018. pp.: 14 - 14.

(MARTECH'16) García, Ó.; Dañobeita, J.; Sorribas, J.; Casas, R.; Piera, J.; Bartolomé, R.; Bardají, R.; Del Rio, J.; Cadena-Muñoz, F.J.; Toma, D.M.; Bghiel, I.; **Martínez, E.**; Nogueras, M. Data acquisition system development for EGIM on EMSODEV EU Project. MARTECH 2016: 7th International Workshop on Marine Technology, Barcelona October 26th, 27th and 28th, 2016. 2016. pp.: 110 - 113.

Bibliography

[1] J. Hildebrand, "Anthropogenic and natural sources of ambient noise in the ocean," *Marine Ecology Progress Series*, vol. 395, pp. 5–20, 2009.

[2] H. Slabbekoorn, N. Bouton, I. van Opzeeland, A. Coers, C. ten Cate, and A. N. Popper, "A noisy spring: the impact of globally rising underwater sound levels on fish," *Trends in Ecology & Evolution*, vol. 25, no. 7, pp. 419–427, 2010.

[3] H. Bailey, B. Senior, D. Simmons, J. Rusin, G. Picken, and P. M. Thompson, "Assessing underwater noise levels during pile-driving at an offshore windfarm and its potential effects on marine mammals," *Marine Pollution Bulletin*, vol. 60, no. 9, pp. 888 – 897, 2010.

[4] G. Zhang, T. N. Forland, E. Johnsen, G. Pedersen, and H. Dong, "Measurements of underwater noise radiated by commercial ships at a cabled ocean observatory," *Marine Pollution Bulletin*, vol. 153, no. January, p. 110948, 2020.

[5] S. Viola, R. Grammauta, V. Sciacca, G. Bellia, L. Beranzoli, G. Buscaino, F. Caruso, F. Chierici, G. Cuttone, A. D'Amico, V. De Luca, D. Embriaco, P. Favali, G. Giovanetti, G. Marinaro, S. Mazzola, F. Filiciotto, G. Pavan, C. Pellegrino, S. Pulvirenti, F. Simeone, F. Speziale, and G. Riccobene, "Continuous monitoring of noise levels in the Gulf of Catania (Ionian Sea). Study of correlation with ship traffic," *Marine Pollution Bulletin*, vol. 121, no. 1-2, pp. 97–103, 2017.

[6] N. D. Merchant, P. Blondel, D. T. Dakin, and J. Dorocicz, "Averaging underwater noise levels for environmental assessment of shipping," *The Journal of the Acoustical Society of America*, vol. 132, no. 4, pp. EL343–EL349, 2012.

[7] P. Madsen, M. Wahlberg, J. Tougaard, K. Lucke, and P. Tyack, "Wind turbine underwater noise and marine mammals: implications of current knowledge and data needs," *Marine Ecology Progress Series*, vol. 309, pp. 279–295, mar 2006.

[8] S. Patrício, A. Moura, and T. Simas, "Wave energy and underwater noise: State of art and uncertainties," in *OCEANS '09 IEEE Bremen: Balancing Technology with Future Needs*, 2009.

[9] The European Parliament and the Council of the European Union, "Marine Strategy Framework Directive," 2008.

[10] A. J. Van der Graaf, M. A. Ainslie, M. André, K. Brensing, J. Dalen, R. P. A. Dekeling, S. Robinson, M. L. Tasker, F. Thomsen, and S. Werner, "European Marine Strategy Framework Directive Good Environmental Status (MSFD-GES): Report of the Technical Subgroup on Underwater noise and other forms of energy.," Tech. Rep. February, Publications Office of the European Union, Brussels, Belgium, 2012.

[11] R. Dekeling, M. Tasker, A. Van der Graaf, M. Ainslie, M. Andersson, M. André, J. Borsani, K. Brensing, M. Castellote, D. Cronin, J. Dalen, T. Folegot, R. Leaper, J. Pajala, P. Redman, S. Robinson, P. Sigray, G. Sutton, F. Thomsen, S. Werner, D. Wittekind, and J. Young, "Monitoring Guidance for Underwater Noise in European Seas, Part II: Monitoring Guidance Specifications," tech. rep., Publications Office of the European Union, Luxembourg, 2014.

[12] J. L. Miksis-Olds and S. M. Nichols, "Is low frequency ocean sound increasing globally?," *The Journal of the Acoustical Society of America*, vol. 139, no. 1, pp. 501–511, 2016.

[13] N. D. Merchant, K. L. Brookes, R. C. Faulkner, A. W. J. Bicknell, B. J. Godley, and M. J. Witt, "Underwater noise levels in UK waters," *Scientific Reports*, vol. 6, p. 36942, dec 2016.

[14] M. van der Schaar, M. A. Ainslie, S. P. Robinson, M. K. Prior, and M. André, "Changes in 63Hz third-octave band sound levels over 42months recorded at four deep-ocean observatories," *Journal of Marine Systems*, vol. 130, pp. 4–11, 2014.

[15] M. Mustonen, A. Klauson, M. Andersson, D. Clorennec, T. Folegot, R. Koza, J. Pajala, L. Persson, J. Tegowski, J. Tougaard, M. Wahlberg, and P. Sigray, "Spatial and Temporal Variability of Ambient Underwater Sound in the Baltic Sea," *Scientific Reports*, vol. 9, no. 1, p. 13237, 2019.

[16] L. Bittencourt, R. R. Carvalho, J. Lailson-Brito, and A. F. Azevedo, "Underwater noise pollution in a coastal tropical environment," *Marine Pollution Bulletin*, vol. 83, no. 1, pp. 331–336, 2014.

[17] J. K. Garrett, P. Blondel, B. J. Godley, S. K. Pikesley, M. J. Witt, and L. Johanning, "Long-term underwater sound measurements in the shipping noise indicator bands 63 Hz and 125 Hz from the port of Falmouth Bay, UK," *Marine Pollution Bulletin*, vol. 110, no. 1, pp. 438–448, 2016.

[18] N. D. Merchant, T. R. Barton, P. M. Thompson, E. Pirotta, D. T. Dakin, and J. Dorocicz, "Spectral probability density as a tool for ambient noise analysis," *The Journal of the Acoustical Society of America*, vol. 133, no. 4, pp. EL262–EL267, 2013.

[19] S. P. Robinson, P. A. Lepper, and R. A. Hazelwood, "Good Practice Guide for Underwater Noise Measurement," Tech. Rep. 133, National Measurement Office, Marine Scotland, The Crown Estate, Teddington, England, 2014.

[20] J. Del-Rio, M. Nogueras, D. M. Toma, E. Martinez, C. Artero-Delgado, I. Bghiel, M. Martinez, J. Cadena, A. Garcia-Benadi, D. Sarria, J. Aguzzi, I. Masmitja, M. Carandell, J. Olive, S. Gomariz, P. Santamaria, and A. Manuel Lazaro, "Obsea: A Decadal Balance for a Cabled Observatory Deployment," *IEEE Access*, vol. 8, pp. 33163–33177, 2020.

[21] T. Daniel, J. Manley, and N. Trenaman, "The Wave Glider: enabling a new approach to persistent ocean observation and research," *Ocean Dynamics*, vol. 61, pp. 1509–1520, oct 2011.

[22] M. H. Ghani, L. R. Hole, I. Fer, V. H. Kourafalou, N. Wienders, H. Kang, K. Drushka, and D. Peddie, "The SailBuoy remotely-controlled unmanned vessel: Measurements of near surface temperature, salinity and oxygen concentration in the Northern Gulf of Mexico," *Methods in Oceanography*, vol. 10, pp. 104–121, sep 2014.

[23] H. Claustre and L. Beguery, "SeaExplorer Glider Breaks Two World Records," *Sea Technology*, vol. 55, no. 3, pp. 19–21, 2014.

[24] M. André, M. van der Schaar, S. Zaugg, L. Houégnigan, A. Sánchez, and J. Castell, "Listening to the Deep: Live monitoring of ocean noise and cetacean acoustic signals," *Marine Pollution Bulletin*, vol. 63, pp. 18–26, jan 2011.

[25] H. R. Kolar, E. P. McKeown, M. E. Purcell, P. J. Gaughan, A. G. Westbrook, M. G. Barry, A. M. Akhriev, J. P. Hayes, A. Castelfranco, G. Nolan, D. J. Murray, K. P. Adlum, and D. F. Glynn, "The design and deployment of a real-time wide spectrum acoustic monitoring system for the ocean energy industry," in *2013 MTS/IEEE OCEANS - Bergen*, 2013.

[26] J. Tegowski, R. Koza, I. Pawliczka, K. Skóra, K. Trzcińska, and J. Zdroik, "Statistical, spectral and wavelet features of the ambient noise detected in the Southern Baltic sea," in *23rd International Congress on Sound and Vibration*, International Institute of Acoustics and Vibration (IIAV), 2016.

[27] C. Jiang, J. Li, and W. Xu, "The Use of Underwater Gliders as Acoustic Sensing Platforms," *Applied Sciences*, vol. 9, p. 4839, nov 2019.

[28] J. Sun, J. Wang, Y. Shi, F. Hu, X. Wang, J. Yu, and A. Zhang, "Self-Noise Spectrum Analysis and Joint Noise Filtering for the Sea-Wing Underwater Glider Based on Experimental Data," *IEEE Access*, vol. 8, pp. 42960–42970, 2020.

[29] L. Liu, L. Xiao, S. Q. Lan, T. T. Liu, and G. L. Song, "Using Petrel II Glider to Analyze Underwater Noise Spectrogram in the South China Sea," *Acoustics Australia*, vol. 46, pp. 151–158, 2018.

[30] A. Dassatti, M. van der Schaar, P. Guerrini, S. Zaugg, L. Houegnigan, A. Maguer, and M. Andre, "On-board underwater glider real-time acoustic environment sensing," in *OCEANS 2011 IEEE - Spain*, IEEE, 2011.

[31] D. M. Toma, I. Masmitja, J. del Río, E. Martinez, C. Artero-Delgado, A. Casale, A. Figoli, D. Pinzani, P. Cervantes, P. Ruiz, S. Memè, and E. Delory, "Smart embedded passive acoustic devices for real-time hydroacoustic surveys," *Measurement*, vol. 125, pp. 592–605, 2018.

[32] J. Del Rio, D. M. Toma, T. C. O'Reilly, A. Broring, D. R. Dana, F. Bache, K. L. Headley, A. Manuel-Lazaro, and D. R. Edgington, "Standards-based plug & work for instruments in ocean observing systems," *IEEE Journal of Oceanic Engineering*, vol. 39, no. 3, pp. 430–443, 2014.

[33] B. M. Howe, Y. Chao, Y. Chao, P. Arabshahi, T. McGinnis, S. Roy, and A. Gray, "A Smart Sensor Web for Ocean Observation: Fixed and Mobile Platforms, Integrated Acoustics, Satellites and Predictive Modeling," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 3, no. 4, pp. 507–521, 2010.

[34] "ISO/IEC/IEEE 24765: Systems and software engineering - Vocabulary," tech. rep., IEEE, New York, USA, 2017.

[35] A. Bröring, J. Echterhoff, S. Jirka, I. Simonis, T. Everding, C. Stasch, S. Liang, and R. Lemmens, "New generation Sensor Web Enablement," *Sensors*, vol. 11, no. 3, pp. 2652–2699, 2011.

[36] C. Petrioli, R. Petroccia, D. Spaccini, A. Vitaletti, T. Arzilli, D. Lamanna, A. Galizial, and E. Renzi, "The SUNRISE GATE: Accessing the SUNRISE federation of facilities to test solutions for the Internet of Underwater Things," in *2014 Underwater Communications and Networking, UComms 2014*, 2014.

[37] C. Reed, M. Botts, J. Davidson, and G. Percivall, "OGC Sensor Web Enablement: Overview and High Level Architecture," *GeoSensor networks*, vol. 4540, pp. 175–190, 2008.

[38] A. Bröring, J. Echterhoff, S. Jirka, I. Simonis, T. Everding, C. Stasch, S. Liang, and R. Lemmens, "New generation Sensor Web Enablement," *Sensors*, vol. 11, no. 3, pp. 2652–2699, 2011.

[39] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J. W. Boiten, L. B. da Silva Santos, P. E. Bourne, J. Bouwman, A. J. Brookes, T. Clark, M. Crosas, I. Dillo, O. Dumon, S. Edmunds, C. T. Evelo, R. Finkers, A. Gonzalez-Beltran, A. J. Gray, P. Groth, C. Goble, J. S. Grethe, J. Heringa, P. A. t Hoen, R. Hooft, T. Kuhn, R. Kok, J. Kok, S. J. Lusher, M. E. Martone, A. Mons, A. L. Packer, B. Persson, P. Rocca-Serra, M. Roos, R. van Schaik, S. A. Sansone, E. Schultes, T. Sengstag, T. Slater, G. Strawn, M. A. Swertz, M. Thompson, J. Van Der Lei, E. Van Mulligen, J. Velterop, A. Waagmeester, P. Wittenburg, K. Wolstencroft, J. Zhao, and B. Mons, "The FAIR Guiding Principles for scientific data management and stewardship," *Scientific Data*, vol. 3, 2016.

[40] T. Tanhua, S. Pouliquen, J. Hausman, K. M. O'Brien, P. Bricher, T. de Bruin, J. J. Buck, E. F. Burger, T. Carval, K. S. Casey, S. Diggs, A. Giorgetti, H. Glaves, V. Harscoat, D. Kinkade, J. H. Muelbert, A. Novellino, B. G. Pfeil, P. Pulsifer, A. P. Van de Putte, E. Robinson, D. Shaap, A. Smirnov, N. Smith, D. P. Snowden, T. Spears, S. Stall, M. Tacoma, P. Thijsse, S. Tronstad, T. Vandenberghe, M. Wengren, L. Wyborn, and Z. Zhao, "Ocean FAIR data services," *Frontiers in Marine Science*, vol. 6, p. 440, 2019.

[41] K. Walter and E. Nash, "Coupling Wireless Sensor Networks and the Sensor Observation Service – Bridging the Interoperability Gap," in *12th AGILE International Conference on Geographic Information Science, Hannover, Germany, 02-05 June*, 2009.

[42] E. Y. Song and K. Lee, "Understanding IEEE 1451 - Networked smart transducer interface standard - What is a smart transducer?," *IEEE Instrumentation and Measurement Magazine*, vol. 11, no. 2, pp. 11–17, 2008.

[43] G. Gigan and I. Atkinson, "Sensor Abstraction Layer: A unique software interface to effectively manage sensor networks," in *Intelligent Sensors, Sensor Networks and Information Processing, ISSNIP, Melbourne, Australia, December 3-6*, pp. 479–484, 2007.

[44] J. Trevathan, R. Johnstone, T. Chiffings, I. Atkinson, N. Bergmann, W. Read, S. Theiss, T. Myers, and T. Stevens, "SEMAT - The next generation of inexpensive marine environmental monitoring and measurement systems," *Sensors (Switzerland)*, vol. 12, no. 7, pp. 9711–9748, 2012.

[45] S. M. Fairgrieve, J. A. Makuch, and S. R. Falke, "PULSENet: An implementation of sensor web standards," in *2009 International Symposium on Collaborative Technologies and Systems, Baltimore, USA, May 18 - 22*, pp. 64–75, 2009.

[46] J. Geipel, M. Jackenkroll, M. Weis, and W. Claupein, "A Sensor Web-Enabled Infrastructure for Precision Farming," *ISPRS International Journal of Geo-Information*, vol. 4, no. 1, pp. 385–399, 2015.

[47] A. Broering, S. Below, and T. Foerster, "Declarative sensor interface descriptors for the sensor web," in *WebMGS 2010: 1st International Workshop on Pervasive Web Mapping, Geoprocessing and Services, Como, Italy, August 26- 27*, pp. 26–32, 2010.

[48] A. Bröring, P. Maué, K. Janowicz, D. Nüst, and C. Malewski, "Semantically-Enabled Sensor Plug & Play for the Sensor Web," *Sensors*, vol. 11, no. 8, pp. 7568–7605, 2011.

[49] A. Bröring, F. Bache, T. Bartoschek, and C. P. J. M. van Elzakker, "The SID Creator: A Visual Approach for Integrating Sensors with the Sensor Web," *Lecture Notes in Geoinformation and Cartography*, pp. 143–162, 2011.

[50] L. Díaz, A. Bröring, D. McInerney, G. Libertá, and T. Foerster, "Publishing sensor observations into Geospatial Information Infrastructures: A use case in fire danger assessment," *Environmental Modelling and Software*, vol. 48, pp. 65–80, 2013.

[51] M. A. Jazayeri, S. H. L. Liang, and C. Y. Huang, "Implementation and evaluation of four interoperable open standards for the internet of things," *Sensors (Switzerland)*, vol. 15, no. 9, pp. 24343–24373, 2015.

[52] S. A. Sawant, J. Adinarayana, and S. S. Durbha, "KrishiSense: A semantically aware web enabled wireless sensor network system for precision agriculture appli-

cations," in *International Geoscience and Remote Sensing Symposium (IGARSS), Quebec City, Canada, July 13-18*, pp. 4090–4093, 2014.

[53] N. Chen, X. Zhang, and C. Wang, "Integrated open geospatial web service enabled cyber-physical information infrastructure for precision agriculture monitoring," *Computers and Electronics in Agriculture*, vol. 111, pp. 78–91, 2015.

[54] A. Kotsev, F. Pantisano, S. Schade, and S. Jirka, "Architecture of a service-enabled sensing platform for the environment," *Sensors (Switzerland)*, vol. 15, no. 2, pp. 4470–4495, 2015.

[55] A. Kotsev, S. Schade, M. Craglia, M. Gerboles, L. Spinelle, and M. Signorini, "Next generation air quality platform: Openness and interoperability for the internet of things," *Sensors (Switzerland)*, vol. 16, no. 3, 2016.

[56] T. O'Reilly, "OGC® PUCK Protocol Standard Version 1.4," tech. rep., Open Geospatial Consortium, Wayland, MA, 01778, USA, 2012.

[57] A. Robin, "OGC SWE Common Data Model Encoding Standard," tech. rep., Open Geospatial Consortium, Wayland, MA, USA, 2011.

[58] M. Botts and A. Robin, "OGC SensorML: Model and XML Encoding Standard 2.0," tech. rep., Open Geospatial Consortium, Wayland, MA, USA, 2014.

[59] T. Paolo, F. Cristiano, O. Alessandro, and C. Paola, "Semantic Profiles for Easing SensorML Description: Review and Proposal," *ISPRS International Journal of Geo-Information*, vol. 8, p. 340, 2019.

[60] S. Cox, "Observations and Measuremets-XML Implementation," tech. rep., Open Geospatial Consortium, Wayland, MA, USA, 2011.

[61] A. Bröring, C. Stasch, and J. Echterhoff, "OGC Sensor Observation Service," tech. rep., Open Geospatial Consortium, Wayland, MA, USA, 2012.

[62] J. Schneider, T. Kamiya, D. Peintner, and R. Kyusakov, "Efficient XML Interchange (EXI) Format 1.0." https://www.w3.org/TR/exi/. Accessed 2021-05-12.

[63] "52° North Sensor Observation Service (SOS)." https://github.com/52North/SOS. Accessed: 2021-05-12.

[64] S. Liang, C.-Y. Huang, and T. Khalafbeigi, "OGC SensorThings API Part 1: Sensing," 2016.

[65] A. Kokkinaki, L. Darroch, J. Buck, and S. Jirka, "Semantically Enhancing SensorML with Controlled Vocabularies in the Marine Domain," in *Proceedings of the Geospatial Sensor Webs Conference*, 2016.

[66] U. K. Verfuß, M. Andersson, T. Folegot, J. Laanearu, R. Matuschek, J. Pajala, P. Sigray, J. Tegowski, and J. Tougaard, "Bias Standards for noise measurements. Background information, Guidelines and Quality Assurance," tech. rep., BIAS Project, 2015.

[67] T. J. Rouphael, *RF and Digital Signal Processing for Software-Defined Radio*. Oxford, UK: Elsevier, 1 ed., 2009.

[68] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing*. Upper Saddle River, NJ, USA: Pearson Prentice Hall, 4 ed., 2007.

[69] F. Harris, "On the use of windows for harmonic analysis with the discrete Fourier transform," *Proceedings of the IEEE*, vol. 66, no. 1, pp. 51–83, 1978.

[70] G. Heinzel, A. Rüdiger, and R. Schilling, "Spectrum and Spectral Density Estimation by the Discrete Fourier Transform(DFT), Including a Comprehensive List of Window Functions and Some New At-Top Windows," tech. rep., Max-Planck-Institut fur Gravitationsphysik, Hannover, Germany, 2002.

[71] P. Welch, "The use of fast Fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms," *IEEE Transactions on Audio and Electroacoustics*, vol. 15, no. 2, pp. 70–73, 1967.

[72] K. Betke, T. Folegot, R. Matuschek, J. Pajala, L. Persson, J. Tegowski, and M. Wahlberg, "BIAS Standards for Signal Processing. Aims, Processes and Recommendations. Amended version," tech. rep., BIAS Project, 2015.

[73] D. Gillespie, D. K. Mellinger, J. Gordon, D. McLaren, P. Redmond, R. McHugh, P. Trinder, X. Deng, and A. Thode, "PAMGUARD: Semiautomated, open source software for real-time acoustic detection and localization of cetaceans.," *The Journal of the Acoustical Society of America*, vol. 125, pp. 2547–2547, apr 2009.

[74] Acoustical Society of America, "ANSI S1.11-2004: Specification for Octave, Half-Octave, and Third Octave Band Filter Sets," tech. rep., American National Standards Institute, Melville, NY, 2004.

[75] N. Lanteri, J. Legrand, B. Moreau, J. R. Lagadec, and J. F. Rolin, "The EGIM, a generic instrumental module to equip EMSO observatories," in *OCEANS 2017 - Aberdeen*, 2017.

[76] P. Favali, J. Dañobeitia, L. Beranzoli, J.-F. Rolin, V. Lykousis, H. A. Ruhl, G. Paul, J. Piera, R. Huber, J. del Río, O. Llinás, J. M. de Miranda, P. Terrinha, V. Radulescu, and N. O'Neill, "European Multidisciplinary and Water-Column Observa tory - European Research Infrastructure Consortium (EMSO ERIC): Challenges and opportunitie s for Strategic European Marine Sciences," in *7th International Workshop on Marine Technology, Barcelona, Spain, October 26-28th, 2016*, pp. 100–103, 2016.

[77] Zabbix LLC, "Zabbix." https://www.zabbix.com. Accessed 2021-05-12.

[78] J. Aguzzi, A. Mànuel, F. Condal, J. Guillén, M. Nogueras, J. del Rio, C. Costa, P. Menesatti, P. Puig, F. Sardà, D. Toma, and A. Palanques, "The new seafloor observatory (OBSEA) for remote and long-term coastal ecosystem monitoring," *Sensors*, vol. 11, no. 6, pp. 5850–5872, 2011.

[79] A. Novellino, P. Gorringe, D. Schaap, S. Pouliquen, L. Rickards, and G. Manzella, "European marine observation data network - EMODnet Physics," in *2014 IEEE/OES Baltic International Symposium (BALTIC)*, IEEE, 2014.

[80] S. Memè, E. Delory, M. Felgines, J. Pearlman, F. Pearlman, J. del Rio, E. Martinez, I. Masmitja, J. Gille, J.-F. Rolin, and Others, "NeXOS—Next generation, cost-effective, compact, multifunctional web enabled ocean sensor systems," in *OCEANS 2017-Anchorage*, IEEE, 2017.

[81] D. M. Toma, J. Del Rio, S. Jirka, E. Delory, J. Pearlman, and C. Waldmann, "NeXOS smart electronic interface for sensor interoperability," in *MTS/IEEE OCEANS 2015: Discovering Sustainable Ocean Energy for a New World, Genova, Italy, May 18-21*, 2015.

[82] E. Delory, E. Martinez, D. M. Toma, J. Del Rio, E. Caudet, C. Waldmann, C. Barrera, J. Danobeitia, and O. Llinás, "Towards Web-Enabled Ocean Sensor Networks," in *AGU Fall Meeting 2019*, AGU, 2019.

[83] D. A. R. Arce, E. Abdi, E. M. Padró, E. Delory, C. B. Solano, J. Hernández, and O. Llinas, "Smart sensing interoperability platforms in the scope of Atlantos," *Instrumentation ViewPoint*, no. 20, p. 32, 2018.

[84] D. M. Toma, C. Artero, J. Del Río, E. Trullols, and X. Roset, "Near Real-Time Determination of Earthquake Source Parameters from the Coastal Ocean," in *7th International Workshop on Marine Technology, Barcelona, Spain, October 26-28th*, (Barcelona), 2016.

[85] X. Roset, E. Trullols, C. Artero-Delgado, J. Prat, J. Del Río, I. Massana, M. Carbonell, J. Barco de la Torre, and D. Toma, "Real-Time Seismic Data from the Bottom Sea," *Sensors*, vol. 18, p. 1132, apr 2018.

Appendix A

Article Compendium

A.1 Article 1

Middleware for Plug and Play Integration of Heterogeneous Sensor Resources into the Sensor Web

---

Journal: *Sensors (Basel)*
Publisher: MDPI
Date of Publication: 15 December 2017
Impact Factor: 3.275
JCR Rank: Q1 Instrumentation / Q2 Electronics

DOI: 10.3390/s17122923

---

sensors

MDPI

Article

Middleware for Plug and Play Integration of Heterogeneous Sensor Resources into the Sensor Web

Enoc Martínez [1,*], Daniel M. Toma [1], Simon Jirka [2] and Joaquín Del Río [1]

[1] SARTI research group, Electronics Department, Universitat Politècnica de Catalunya, 08800 Vilanova i la Geltrú, Spain; daniel.mihai.toma@upc.edu (D.M.T.); joaquin.del.rio@upc.edu (J.D.R.)
[2] 52° North Initiative for Geospatial Open Source Software, 48151 Münster, Germany; jirka@52north.org
* Correspondence: enoc.martinez@upc.edu; Tel.: +34-93-896-7205

Received: 6 November 2017; Accepted: 8 December 2017; Published: 15 December 2017

Abstract: The study of global phenomena requires the combination of a considerable amount of data coming from different sources, acquired by different observation platforms and managed by institutions working in different scientific fields. Merging this data to provide extensive and complete data sets to monitor the long-term, global changes of our oceans is a major challenge. The data acquisition and data archival procedures usually vary significantly depending on the acquisition platform. This lack of standardization ultimately leads to information silos, preventing the data to be effectively shared across different scientific communities. In the past years, important steps have been taken in order to improve both standardization and interoperability, such as the Open Geospatial Consortium's Sensor Web Enablement (SWE) framework. Within this framework, standardized models and interfaces to archive, access and visualize the data from heterogeneous sensor resources have been proposed. However, due to the wide variety of software and hardware architectures presented by marine sensors and marine observation platforms, there is still a lack of uniform procedures to integrate sensors into existing SWE-based data infrastructures. In this work, a framework aimed to enable sensor plug and play integration into existing SWE-based data infrastructures is presented. First, an analysis of the operations required to automatically identify, configure and operate a sensor are analysed. Then, the metadata required for these operations is structured in a standard way. Afterwards, a modular, plug and play, SWE-based acquisition chain is proposed. Finally different use cases for this framework are presented.

Keywords: Sensor Web Enablement; plug and play; interoperability; SensorML; Open Geospatial Consortium; sensor integration; OGC PUCK protocol

1. Introduction

As the interest of studying global environmental phenomena is growing, collaborative research environments are becoming more important. In these environments, data coming from many different sources has to be combined and managed in a standard way in order to avoid information silos. The lack of standardization and data harmonization across scientific domains and scientific data infrastructures has been the driving force for the Open Geospatial Consortium (OGC) to propose the Sensor Web Enablement framework (SWE) [1]. This framework is a suite of data model, encoding, and interface standards which aim to provide the building blocks for interoperable Sensor Web infrastructures. In this context, the concept of the Sensor Web refers to a set of Web accessible sensor networks and their collected sensor data/metadata that can be discovered and accessed using standard protocols and application programming interfaces [2].

The different components in an SWE-based architecture can be classified according to their role in a Sensor Web layer stack [3]. In Figure 1 a Sensor Web layer stack for ocean observing systems is presented. Although the interaction patterns in the upper layers of the SWE stack are strictly specified by the SWE standards, integrating sensors into Sensor Web services is not fully defined: While the transactional operations of the OGC Sensor Observation Service already allow the standardized publication of data, certain aspects such as sensor discovery and plug and play mechanisms are not yet sufficiently available. Therefore, bridging between the physical sensors and the Sensor Web services is still a challenge [4]. Many SWE-based architectures currently in operation still rely on proprietary mechanisms to integrate sensor data streams (e.g., by customizing SWE services to use existing observation databases as data source).

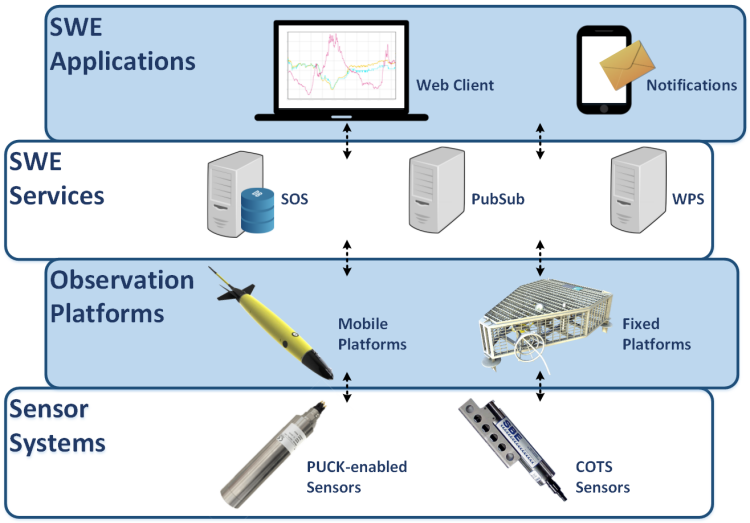


Figure 1. Sensor Web Enablement layer stack for ocean observing systems. The sensor layer comprises the physical sensors. The integration layer includes all the mechanisms that bridge the data from the sensor's output to Sensor Web services. The Sensor Web Services layer is the core of a Sensor Web infrastructure, where the data is archived, processed and analysed. Finally the application layer provides interfaces between the Sensor Web services and the final users (e.g., data viewers).

In ocean observing systems, instruments and sensor systems (in short: sensors) are commonly deployed using observation platforms, which are in charge of operating sensors and acquiring their data. The ocean observing community uses a vast collection of observation platforms, such as fixed underwater observatories, buoys, underwater gliders, autonomous surface vehicles, profilers, etc. Some of them are powered by batteries and use satellite communication links (i.e., underwater gliders), while others may be connected to the electrical grid and use broadband Ethernet (underwater cabled observatories). Due to their heterogeneous nature, observation platforms present a wide variety of architectures, resulting in a wide range of non-standardized protocols, data encodings, and communication interfaces.

Due to this heterogeneity, in most cases a specific driver is needed for each sensor-platform combination in order to gather data. Sensor manufacturers usually provide drivers for desktop environments (i.e., Microsoft Windows). However, marine sensors are frequently integrated in embedded environments where the provided drivers cannot be used [5]. If a new sensor has to be integrated or an existing sensor has to be moved from one platform to another, new specific components have to be developed. Generating a specific driver for each sensor is a time-consuming task that requires in-depth knowledge of both sensor's protocol and the observation platform's architecture [5].

Moreover, when integrating several platforms into SWE-based data infrastructures, each platform-specific encoding needs to be adapted by service adapters. This conversion can either be achieved directly on the platform or by an intermediate process on a land station server in the case of resource-constrained platforms. Figure 2 depicts a non-standardized scenario where different sensors are deployed using different observation platforms. As the number of platforms and sensors deployed within a collaborative environment grows, the number of custom components increases, as well as the infrastructure maintenance costs. Therefore, improving the interoperability among the infrastructure's components should be a matter of great concern. Interoperability in this sense can be defined as the ability of two systems to exchange information and to interpret the information that has been exchanged [6].

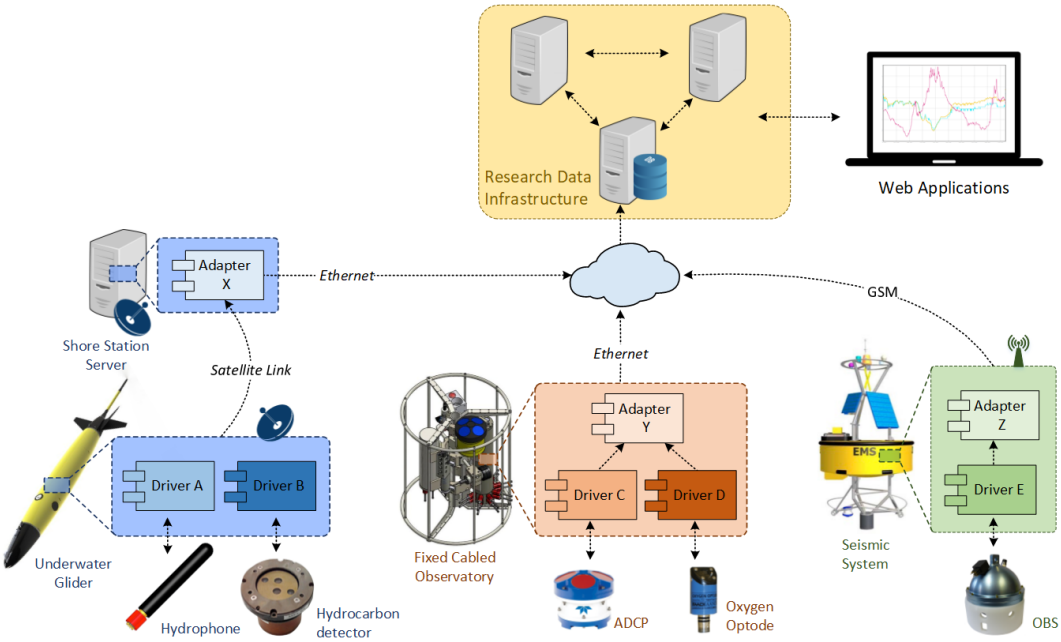


Figure 2. Collaborative research scenario using specific drivers and data converters for each sensor-platform combination, in this case an underwater glider, a buoy and a cabled underwater observatory.

The ultimate goal of interoperability in a collaborative research environment should be to enable the plug and play integration of sensor resources into observation platforms and data infrastructures, reducing the human intervention to a minimum. This goal implies two different levels of interoperability: operational and data management. From the operational interoperability point of view, a sensor should be automatically detected, configured and its data gathered by the observation platform's acquisition process, as soon as it is deployed into an observation platform. From the data management side, the data retrieved from the sensor should be encoded in a standard format, compatible with the data infrastructure. Sensor metadata plays a key role in the acquisition. If managed properly it will enhance not only the interoperability of the sensor, but also its data traceability and ease further data quality procedures, allowing to pinpoint for example sensor malfunctions [7]. Therefore, metadata should be added alongside sensor data, providing contextual information [8].

Within this work, a standardized, plug and play acquisition chain capable of bridging sensor resources to Sensor Web services is envisioned, implemented and evaluated. Instead of using a specific driver for each sensor-platform combination, the use of a generic driver component configurable through OGC standards is proposed. In order to abstract the peculiarities of each sensor, this approach makes use of a metadata file with a machine-understandable unambiguous sensor description, encoded in a standard way. Afterwards, the generic driver should be able to access such a sensor description file and configure itself accordingly, being able to operate a sensor without any a priori knowledge

of the device nor the observation platform where it is hosted. The gathered sensor data should be encoded in standardized SWE formats, ensuring the plug and play integration of sensors into SWE services. This generic component should also have its own auto-description methods, in order to make their role understandable to both humans and machines. The envisioned architecture is presented in Figure 3.

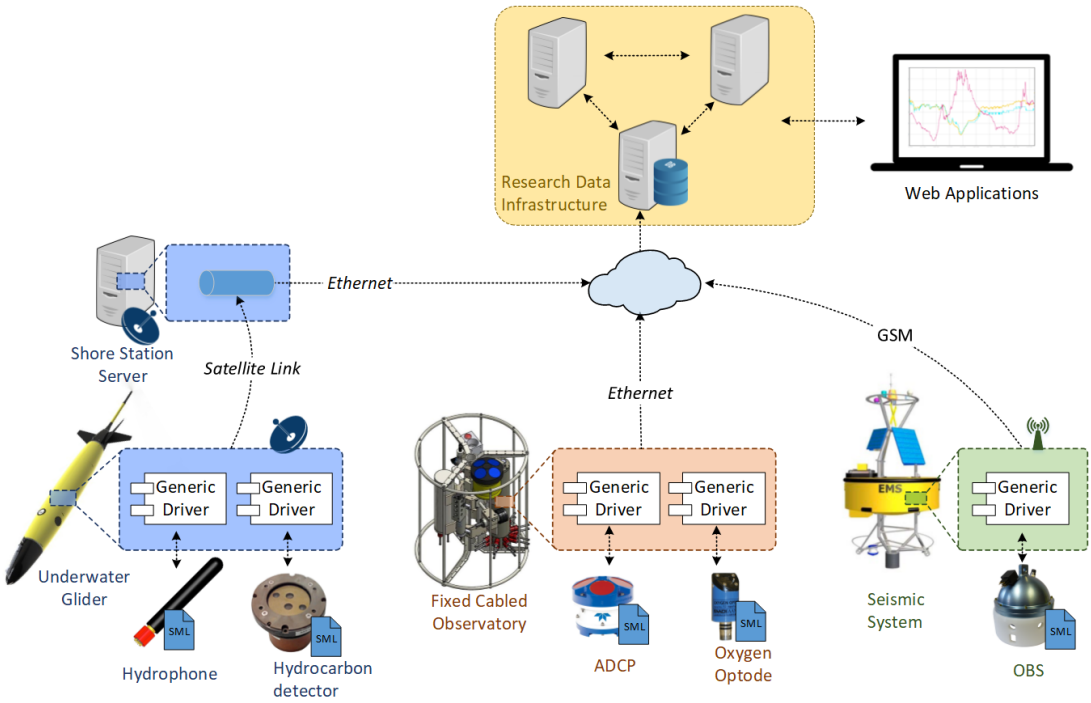


Figure 3. Proposed Sensor Web Enablement (SWE)-based architecture. Sensor resources are integrated by combining a standards-based description with a generic driver in order acquire data. As the generic driver provides a standardized output, the generated data can be directly sent to the SWE services using the observation platform's communication link.

Related Work

The SWE framework aims at tackling a huge interoperability challenge for middleware approaches since heterogeneous devices are expected to collaborate together in communication and information exchange. Implementations of Sensor Web Enablement standards across different domains have found difficulties when integrating sensors into SWE services. Walter and Nash analysed the difficulties of integrating sensors and proposed the use of lightweight SWE connectors (custom drivers) in order to bridge from sensor-specific protocols to SWE services [9]. This approach has been widely used in different applications, using custom drivers or plug-ins as SWE connectors [10,11].

A slightly different approach is to enhance each sensor controller with embedded Web services in order to archive and exchange data in a standardized manner with the components in the upper layers of the Sensor Web Enablement stack. This sensor integration strategy has been used in different fields, such as precision agriculture applications [12] and air quality monitoring [13,14]. This approach, although it provides a good interoperable interface with other Sensor Web-enabled components, does not directly solve the challenge of integrating new sensors into a platform as it still relies on custom drivers.

Another issue that arises when embedding SWE services into sensors controllers, is that SWE services largely rely on exchanging eXstensible Markup Language (XML) files [15], which can be very verbose and require a significant amount of resources to process. When integrating Web services on resource-constrained devices in terms of computational power or communication bandwidth, the use of plain XML may not be suitable. Different strategies on the optimization on the information exchange have been studied in [16]. The Efficient Extensible Interchange format showed promising results in terms of required computational resources and information compactness [17].

To overcome the presented limitations and address these integration challenges, the IEEE 1451 standard proposed the idea of a smart sensor [18]. Smart sensors are defined by the IEEE 1451 standard as sensors with small memory and standardized physical connection to enable the communication with data network and processor. This standard has also been applied to bridge sensors to SWE services [19]. However, this approach assumes an IEEE 1451 compliant sensor device. A broad implementation of this standard has not yet been achieved (the vast majority of commercial sensors do not support IEEE 1451) so that most sensors cannot be integrated using this approach. Moreover, the IEEE 1451 compliant sensor devices are limited by the lack of flexibility, absence of customization options, narrow spectrum of applications, and the basic communication protocol.

A more recent approach is the OGC SensorThings, dedicated especially to Internet of Things (IoT) sensors [20], where sensors are envisioned as web-accessible devices. However, in real world scenarios sensors may be deployed in remote and inaccessible observation platforms where internet connectivity cannot be guaranteed.

As in real world applications a huge variety of sensor protocols (standardized or proprietary) are utilized, another approach is to address the interoperability gap from the opposite direction, by introducing mechanisms to abstract from the variety of sensor protocols, such as Sensor Abstraction Layer (SAL) [21] or Sensor Interface Descriptor (SID) [22].

The Sensor Abstraction Layer (SAL), which describes all the sensors in a uniform manner, makes use of and expands the SensorML 1.0 standard to describe sensor interfaces and commands. It hides the specific technological details by matching sensor-specific commands to generic SAL commands. However, it still relies on a plug-in approach to integrate new sensors [23].

Similar to SAL, the Sensor Interface Descriptor (SID) model extends SensorML 1.0 to formally describe a sensor's protocol. The generated sensor interface description is used as a platform independent sensor driver which contains the necessary information to integrate a sensor on demand by translating between sensor protocol and Sensor Web protocols. A benefit of SID is the availability of open-source components to generate SID descriptions [24] as well as a SID interpreter middleware based on Java. Several applications of SID can be found at the literature [5,25,26].

However, a still remaining open challenge is to practically include such a universal approach in the Sensor Web middleware of marine observatory platforms. A Java-based middleware is a good approach to abstract on the operating system level, however, it may not suitable to be deployed in resource-constrained platforms. Furthermore, the SID model extension defines the whole Open Systems Interconnection (OSI) model stack for each sensor command, resulting in very verbose XML files, which are rather difficult to interpret.

In this work the focus is put on semantic interoperability, emphasizing the need of standard middleware components compatible with different observation platforms. The rest of this paper is structured as follows: Section 2 discusses the requirements to achieve a true plug and play sensor integration into existing data infrastructures. Section 3 presents an SWE-based universal acquisition chain. Section 4 presents Sensor Deployment Files (SDF) as a standard way to organize all the metadata related to a sensor deployment and how it can enhance interoperability. In Section 5 the SWE Bridge middleware is presented, a universal plug and play sensor data acquisition middleware. Finally in Section 6 different use cases of the acquisition chain are presented.

2. Sensor Integration into Observation Platforms

2.1. Requirement Analysis

When integrating new sensor resources into existing data infrastructures, several standardized operations are required. These operations can be classified in two different levels: Instrument Level, and Sensor Web level [5]. The Instrument Level operations are related to the sensor's operational challenges, focused on the sensor-platform interaction, including both sensor configuration and sensor data retrieval. The Sensor Web challenges are related to the sensor data management, including standardized data encodings, sensor data discovery, tasking mechanisms, etc.

According to previous work on this topic, at least four operations are required at the instrument level: sensor detection, sensor identification, sensor configuration and simple measurements operations [5]. These are operational requirements, focused on the direct sensor-platform interaction.

From the data management side, there is a multitude of operations that can be realized, such as data discovery, data access, sensor tasking, events and notifications, etc. However, this work focuses on the integration of sensors to the Sensor Web from a platform operator point of view. Thus only the sensor registration and sensor data ingestion operations are considered within this work.

Some ocean observation platforms present additional constrains such as limited power availability and low-bandwidth communications. These platforms are usually based on low-power embedded systems, with limited computational capacity. Thus, a sensor plug and play mechanism should take into account these constrains. Summarizing, the requirements to achieve plug and play integration into observation platforms are:

1. **Sensor Detection**: Detect a new sensor when it is attached to an observation platform. The host platform controller should be able to detect a new sensor without human intervention.
2. **Sensor Identification**: Obtain an unambiguous description of the sensor, including all the metadata to identify the sensor (unique ID, sensor model, etc.) and all information required to register the sensor to an existing Sensor Web server.
3. **Sensor Configuration**: This requirement addresses all operations required before the platform can start retrieving data from a sensor. This includes establishing a communication link between the platform and the sensor and applying any configuration required by the sensor (i.e., activate a specific acquisition channel, set the sampling rate, etc.).
4. **Simple Measurements Operations**: Those operations that are directly related to the retrieval of data. These operations may be actively querying the sensor for data or listening to data streams. Also knowledge of the data interface provided by the sensor is required in order to parse, process and store the data.
5. **Sensor Registration**: Registering a sensor to existing Sensor Web server requires a considerable amount of metadata organized and structured in a coherent way, including physical parameters that are being measured (observable properties), computational representation of the real-world feature that is being measured (feature of interest), alongside with other sensor characteristics. Furthermore, the meaning of this metadata has to be made explicit and understandable by machines, thus, controlled vocabularies containing formal definitions shall be used [27].
6. **Data Ingestion**: Once the sensor is registered, the data measured by this sensor has to be ingested to the server, where it will be archived.
7. **Resource Constrains**: Any plug and play mechanism aimed to integrate sensors into marine data acquisition platforms should be able to work in low-bandwidth, low-power and computationally-constrained scenarios.

2.2. Protocols and Standards

In order to fulfil the previously presented requirements in a standardized way, a set of protocols and standards suitable of fulfilling these needs are presented in this section. The SWE framework provides a set of standards and protocols that can fulfil these needs, as shown in Figure 4.

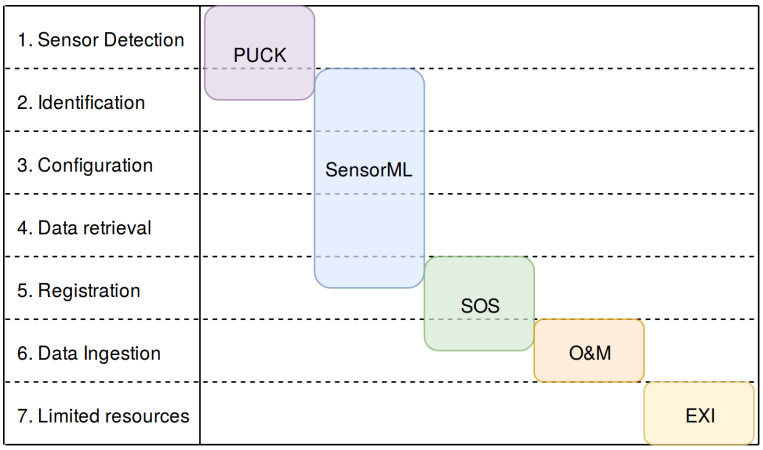


Figure 4. Requirements for a plug and play mechanism (rows) and protocols/standards that can fulfil these requirements (columns).

2.2.1. OGC PUCK Protocol

The sensor detection requirement can be fulfilled using the OGC PUCK protocol within the acquisition platform controller [28]. This protocol is an add-on that can be implemented in any serial or Ethernet sensor alongside with any proprietary protocol, rather than replacing it. This protocol defines a set of commands that grant transparent access to an internal memory, named *OGC PUCK payload*. This payload is frequently used to store sensor metadata. Another key feature of the OGC Puck protocol is its *softbreak* operation, which provides on-the-fly detection without any prior knowledge of the sensor, fulfilling requirement 1.

2.2.2. Sensor Model Language

The OGC Sensor Model Language (SensorML) permits to encode detailed sensor descriptions within an XML file [29]. Its main goal is to enhance interoperability, making sensor descriptions understandable by machines and shareable between intelligent nodes. Moreover, additional information related to specific deployments can also be encoded using this standard. Thus, both sensor configuration and measurement operations can be addressed with the SensorML standard.

It is highly flexible and modular as it can describe almost every sensor related property or sensor-related process. However this flexibility and modularity can prove a double-edged sword, as the same information can be encoded in different ways, increasing the difficulty to generate smart processes capable of interpreting SensorML definitions. For this reason there is ongoing work to develop marine SWE profiles of SensorML that define more precisely how this standard should be applied in ocean observing applications [30].

By combining the OGC PUCK protocol and a SensorML description file, it is possible to automatically detect a sensor and retrieve its description encoded in a single standardized file without any a-priori knowledge about the sensor. If an interpreter software can interpret this file, it can identify the sensor, configure it and retrieve its data, meeting the requirements 2, 3 and 4.

2.2.3. Sensor Observation Service

The SOS standard provides the set of operations required to provide access sensor observation data/metadata as well as to register and archive sensor data and metadata within a data repository [31]. Due to its role as a data and metadata archive, this standard is a key piece of any Sensor Web infrastructure, providing support for requirement 5, sensor registration.

2.2.4. Observations and Measurements

The Observations and Measurements (O&M) standard specifies an abstract model as well as XML encoding for observations and related data, such as features involved in the sampling process [32]. It provides an uniform and unambiguous way to encode sensor measurements, fulfilling the requirement 6.

2.2.5. Efficient XML Interchange

The Efficient XML Interchange (EXI) is a World Wide Web Consortium's (W3C) format that enables the compression of large ASCII XML files into efficient binary files, significantly reducing its size. This format has been designed to allow information sharing between devices with constrained resources, meeting requirement 7 [17].

3. SWE-Based Acquisition Chain

Using the standards and protocols presented in the previous section, an SWE-based, plug and play acquisition chain deployable in a wide variety of ocean observing systems is envisioned. A set of interoperable standard components are proposed in order to bring data from the sensor itself to the Sensor Web, regardless of the constraints of the platform where the sensors are deployed. Figure 5 shows the proposed acquisition chain with its components, classified within the Sensor Web Layer stack. In order to make this architecture suitable for a wide range of scenarios, emphasis has been put on open source software components as well as cross-platform implementations. Each sensor has its own associated Sensor Deployment File (SDF), which encapsulates an unambiguous description of the sensor. The SWE Bridge interfaces the sensor, discovering, operating the sensor and storing its data in standard O&M data files. It also has a SensorML description file, where all its functionalities are described. The files generated as SWE Bridge's output are passed to the SOS Proxy using the observation platform's communication channel (dependent on the acquisition platform). Finally the SOS Proxy injects the O&M data to the SOS server by using its SOS interface. The SOS server archives this data into a SOS database and also provides an interface to access the archived data in a standard manner for further processes, i.e., data visualization web clients.

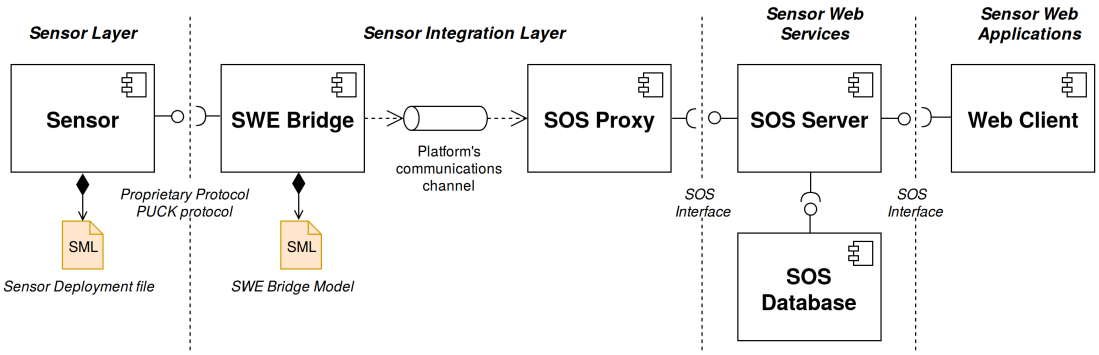


Figure 5. Proposed SWE-based acquisition chain.

3.1. Sensor

Sensors are the component gathering data and making it available through a communication interface, usually serial port or Ethernet. Three different kinds of sensors are taken into account within this work:

- **COTS Sensors**: Commercial off-the-shelf (COTS) sensors are commercially available devices without any particular enhancement in terms of interoperability.
- **OGC PUCK-enabled Sensors**: Sensors which implement the OGC PUCK protocol.

- **Virtual Instruments**: Software components that merge or process data from different sources, generating new data sets accessible through a communication interface.

Each sensor should have its own associated SDF, a SensorML description containing both sensor metadata and deployment information (see Section 4) regardless of their nature. This file is a key piece of the architecture, as it contains all metadata required to describe, operate and register a sensor into Sensor Web services. OGC PUCK-enabled sensors may have their own SDF embedded within its internal payload memory, providing automatic detection and self-description capabilities.

Ideally physical sensors should be OGC PUCK-enabled to provide end to end plug and play capabilities. However, the majority of commercial sensors do not implement this protocol and do not provide auto-detection and self-description procedures. Therefore the operators have to manually match the sensor with SDF stored locally on the platform in order to provide compatibility with the proposed standardized architecture.

3.2. SWE Bridge

The SWE Bridge is a universal acquisition middleware, aimed to be deployed in any acquisition platform, regardless of its software and hardware architecture. It interprets SDFs, automatically configuring itself to connect to and operate a sensor. Its implementation of the OGC PUCK protocol allows to automatically retrieve a SDF from the sensor itself, enhancing interoperability and providing a plug and play mechanism. When interfacing with a COTS a local SDF has to be used. The acquired data is stored in standard O&M files, encoded in XML or EXI format in order to reduce their size [17].

The observation platform's communication link to a shore station can be critical in terms of bandwidth as well as in terms of power consumption. Therefore the SWE Bridge does not directly send the O&M files to the server, but relies on the platform's operator to setup for this transmission. The platform operator can decide under which conditions is desirable to transmit these files (i.e., stopping the transmission when the battery is low). Thus the platform operator still has full control of the platform.

The SWE Bridge has its own SensorML description, the SWE Bridge Model, where all its functionalities, parameters and built-in functions are described in a standard manner. This model allows an automated process to understand the role and the capabilities of the SWE Bridge within the acquisition chain.

3.3. SOS Proxy

The main functionality of the SOS Proxy is to decouple the SOS transactions from constrained platforms without Internet gateway, such as satellite-based platforms. It acts as a transparent intermediary layer, which takes O&M files from the platform's communications channel (i.e., satellite link) and forwards them to an SOS Server. As the SOS Proxy is completely transparent to the rest of the architecture, it does not have an associated SensorML description.

Depending on the nature of the observation platform, the SOS Proxy can be deployed in the platform itself (i.e., a platform with direct Internet access through a GSM modem), otherwise it can be deployed in a shore station server (i.e., platform with proprietary satellite communications). Open source implementations of this software component are available in Java and Bash [33,34].

3.4. SOS Server

The Sensor Observation Service (SOS) acts as a server for storing and managing both sensor data and metadata. Due to its data archiving and metadata management roles, as well as its ability to interface with further services, it is one of the core components of the proposed SWE-based cyber-infrastructure, vital to interact with further processes.

In the proposed acquisition chain the 52° North's open source SOS implementation running on a PostgreSQL database is used [35]. For ingesting metadata about a sensor into an SOS server (registering

a new sensor) the transactional *InsertSensor* operation of the SOS interface is used. The upload of data to the SOS server is achieved through the so called *ResultHandling* operations of the SOS standard (*InsertResultTemplate* and *InsertResult*). To ensure an efficient data transfer the server supports the insertion of EXI-encoded O&M files.

The SOS Server also provides a standardized interface for further process to query and access sensor data archived in the SOS database. SWE services can connect to the SOS server to query for archived sensor data and related metadata.

3.5. Web Client

In order to demonstrate the end-to-end integration a Sensor Web visualization client was developed. For this purpose the 52° North Helgoland Sensor Web viewer is used, which is an open-source, lightweight Web application that enables the exploration, analysis and visualization of sensor web data [36]. To support marine application, Helgoland is designed to support different types of platforms (i.e., stationary and mobile) as well as different types of observation data (e.g., time series, profiles, trajectories).

4. Sensor Deployment Files

The OGC SensorML standard can provide a robust and semantically-tied description of a sensor, including its metadata, communication's interface and command set. However, a sensor can be a complex system configurable in different ways, depending on the deployment and its desired behaviour (i.e., change the sampling rate, select a specific acquisition channel, etc.). All these operations are not only related to the sensor description, but also to the desired acquisition process itself. Thus, alongside the sensor's description there should be a description of the desired acquisition process for each deployment.

In order to enable on-the-fly integration of complex sensor systems into observation platforms, the Sensor Deployment Files (SDF) are introduced. These files, based on the SensorML standard, compile and organize in a coherent manner the sensor's metadata and an accurate description of the desired acquisition process for a specific sensor deployment. They allow an interpreter software, such as the SWE Bridge, to automatically configure the sensor, retrieve its data and store it in standard O&M files. A SDF should contain at least the following information in order to enable plug and play capabilities:

- **Identification**: Provide the required identifiers for the sensor, such as unique ID, model, name, manufacturer, etc. Define which physical parameters is the sensor able to measure.
- **Communications Interface**: An unambiguous and accurate description of the sensor's communication interface to allow an interpreter software to automatically establish a communication link without any a priory information about of the sensor.
- **Communication protocol**: Set of commands required to operate the instrument. This includes configuration commands and measuring operations, as well as a description of the encoding of the sensor outputs.
- **Operation**: Detailed description of the sensor operation, including which operations need to be executed, in which order, which post-processing procedures will be applied to the sensor data and how this data will be stored.

Alongside with this operational information, data management metadata can be included (where the sensor is deployed, in which platform is deployed, etc.). This metadata would allow further processes to interpret a SDF and automatically register the described sensor to a SOS instance without human intervention. This metadata is modelled as optional information in the Sensor Instance and the Sensor Model diagrams. Additional information such as calibration, deployment history or event contact list could also be added to a SDF.

A SDF is composed by a Sensor Instance (sensor's metadata, inherited from a Sensor Model) and a Sensor Mission, as shown in Figure 6. The Sensor Mission uses the descriptions of both the Sensor

Instance and the SWE Bridge Model (see Section 5.2) in order to arrange the available functionalities, defining how the acquisition chain should be configured.

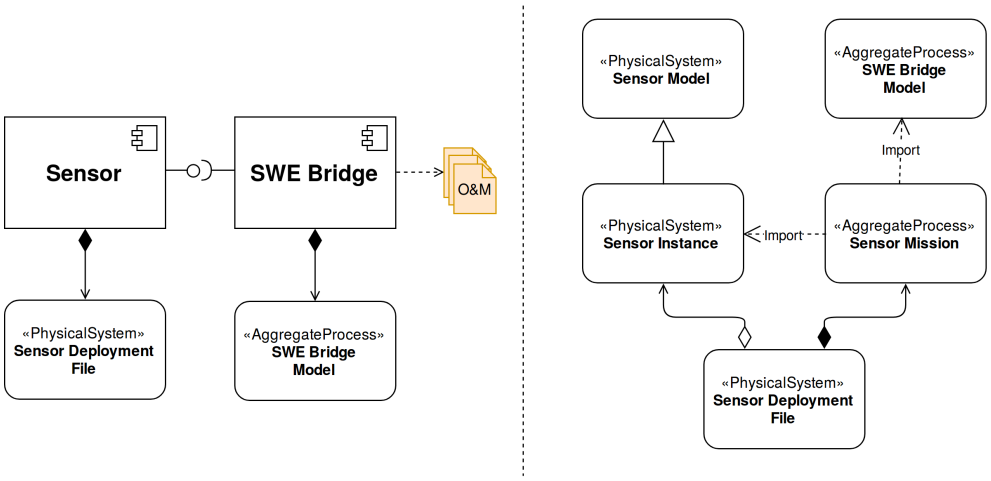


Figure 6. Sensor and SWE Bridge with their models (**left**) and Sensor Deployment Files (SDF) model (**right**). The Sensor, with its associated SDF, is interfaced by the SWE Bridge middleware, which also has its own Sensor Model Language (SensorML)-based description, the SWE Bridge Model.

The potential of SDF is its ability to describe a sensor and configure an acquisition chain with a single, standards-based file. Furthermore, as all the components related to the acquisition chain are described in a formal way, an automated system could process and understand these descriptions and automatically generate new SDF files to setup and update acquisition chains. A set of example SDF can be found at [37].

4.1. Sensor Model

The Sensor Model is a generic SensorML description of a family of sensors having common characteristics. As shown in Figure 7, it should include the sensor's command set, its communications interface and other information applicable to all the sensors that share this model.

The *DataInterface* element is used to model the communication interface. As the interface parameters may depend on each deployment of the sensor, only the available communication protocols are defined at the Sensor Model stage.

A key aspect of a sensor model is the description of the sensor's set of commands. Each command is described with a SensorML's *SimpleProcess* element. At least one *SimpleProcess* is required in a Sensor Model (a sensor should at least provide one operation to retrieve data). The inputs of these processes define the command that the sensor expects, while the output corresponds to the sensor response to that command. If a sensor command does not have a reply to a specific command, the *SimpleProcess* will only have an input and no output. On the contrary, if the sensor streams data periodically the *SimpleProcess* used to model it will not have any input, but will have an output representing this stream. To model a command where the sensor responds to a specific command, both input and output need to be included.

Both input and output have an *encoding* section which describes how their contents (e.g., parameters, output values) are encoded. The output also contains an encoding element alongside with an array of fields, which are used to model the sensor response. Each field has a name, a *description* (reference to a controlled vocabulary) and a *type*, which corresponds to the SWE Common Data model encoding used to model this value [38]. When a physical magnitude is described, the units of measurement should be specified using the *uom* (units of measurement) element.

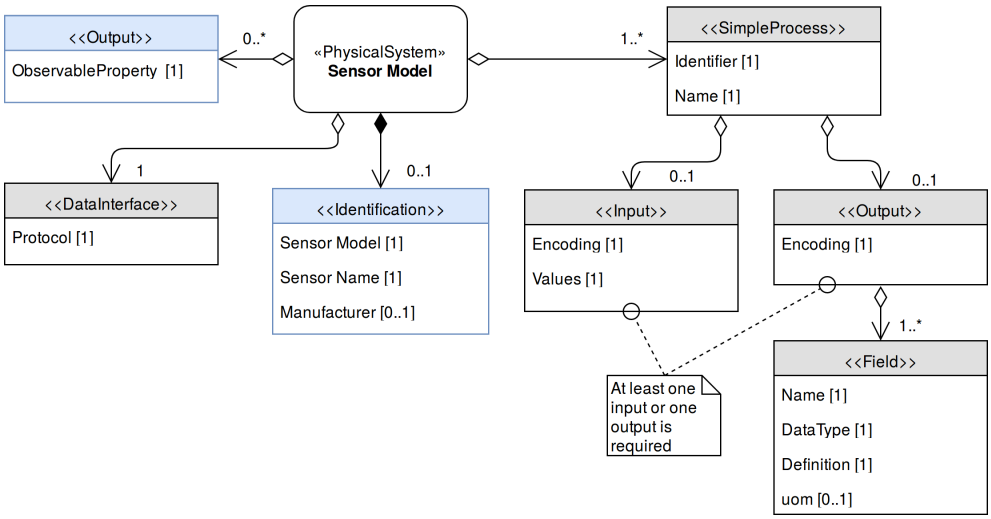


Figure 7. UML diagram of the Sensor Model. The gray elements represent all required information by the SWE Bridge (compulsory) while the blue elements represent optional metadata (used to register the sensor to a Sensor Observation Service (SOS) Instance).

4.2. Sensor Instance

The Sensor Instance models a specific sensor by inheriting a Sensor Model and expanding it with static metadata related to a particular instance (such as unique ID) alongside with dynamic deployment information (such as position). A sensor and its host observation platform may be deployed in remote regions with low bandwidth communication, or no communication link at all. Therefore, a Sensor Instance cannot reference an on-line resource containing its Sensor Model description. Instead it expands a sensor model, resulting in a single file containing all the sensor's metadata. Its model is shown in Figure 8.

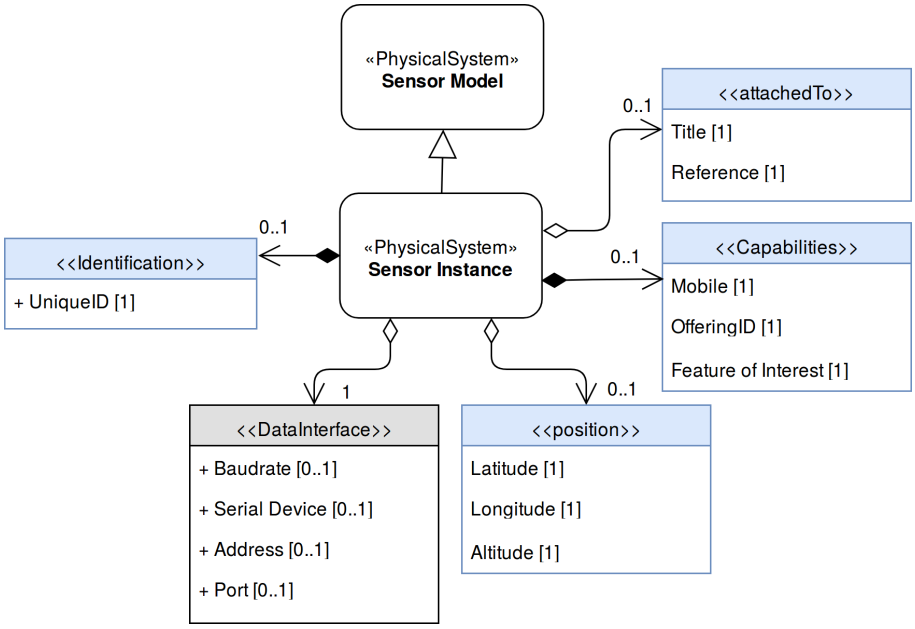


Figure 8. Sensor Instance UML diagram. It inherits a sensor model and expands it with information related to a sensor instance alongside with information related to a specific deployment. The gray elements represent all the required information by the SWE Bridge (compulsory) while the blue elements represent optional metadata (used to register the sensor to a SOS Instance).

From the operational point of view, the more important part of the Sensor Instance is the *DataInterface*. It expands the generic definition of the Sensor Model, defining the parameters required to establish a communication link from the observation platform to the sensor.

In the Sensor Instance model it is possible to include different elements that may be useful to register, discover and exploit the generated data (blue elements). Some of them are the *UniqueID*, *attachedTo* (reference to the observation platform where the sensor is deployed), position (spatial position where the sensor is deployed), *FeatureOfInterest* (a computational representation of the real world phenomenon being observed by the sensor).

4.3. Sensor Mission

The Sensor Mission models the desired acquisition process for a specific sensor deployment. This mission will be interpreted by the SWE Bridge or another implementation of a SDF interpreter, which will setup the acquisition process accordingly. The model of the Sensor Mission is depicted in Figure 9.

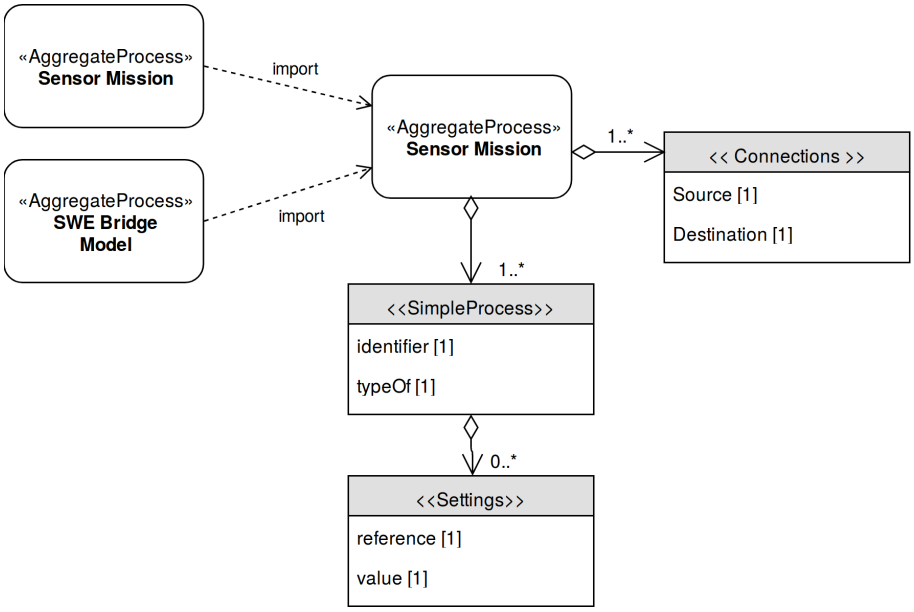


Figure 9. Sensor Mission UML Diagram.

The core of the Sensor Mission is a set of *SimpleProcess* elements which represent the different operations that will be performed by the observation platform's acquisition software. The operations may include data retrieval through sensor commands, post-measurement operations and data storage. Using the *typeOf* property, these *SimpleProcess* can be identified as instances of a specific sensor command (defined in the Sensor Instance) or a built-in SWE Bridge function (defined in the SWE Bridge Model). To allow a flexible configuration, an array of settings may be included, which may modify the default values inherited from the parent process.

The instantiated processes can be connected among them using *connections*, creating chains of processes. These chains of processes contain all operations required by the acquisition process: data retrieval, data processing and data storage. This provides the user a highly flexible framework to configure an acquisition process based on standard SensorML files.

4.4. Sensor Deployment Files and SOS Registration

SDF are mainly focused to address the operational interoperability challenges detected when integrating a new sensor to an observation platform (requirements 2–4, Section 2.1). However, when registering a new sensor resource to a Sensor Observation Service, a different procedure is needed.

Although an SDF contains all the required metadata to register a sensor to an SOS server, it needs to be mapped to the transactional SOS operations.

These operations are *InsertSensor*, which registers the sensor metadata, and *InsertResultTemplate*, which registers the data structure and the encoding of the sensor's observations. If the sensor was previously registered in another deployment in the same SOS server it is possible to modify the sensor metadata by using the *UpdateSensorDescription* operation. The information contained within a SDF has to be mapped to those SOS operations, as shown in Figure 10. After this workflow, a sensor has been registered to an SOS server, so that the upload of the measured data can be started through the *InsertResult* operation.

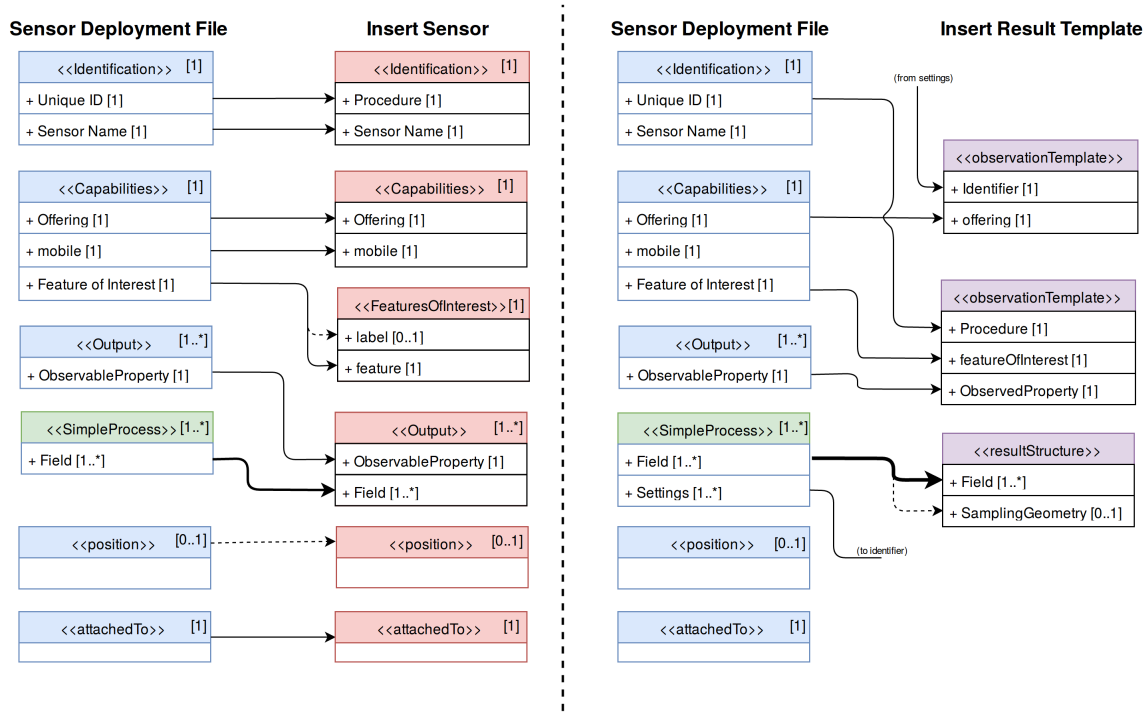


Figure 10. Metadata mapping from Sensor Deployment Files (SDF) to the Sensor Observation Service (SOS) transactional operations *InsertSensor* (**left**) and *InsertResultTemplate* (**right**). The blue elements correspond to Sensor Instance elements while the green elements correspond to Sensor Mission elements. Dashed lines indicate optional elements and thick lines indicate multiple elements.

5. Standards-Based Universal Acquisition Middleware

5.1. Background

In the previous section an approach how to organize sensor metadata into standardized SensorML-encoded SDF has been discussed. However, in order to provide a plug and play framework, a middleware capable of automatically retrieving this SDF, interpret it and configure an acquisition the process is required.

Some marine observation platforms are deployed in long-term missions in remote and inaccessible places (i.e., underwater gliders and profilers). Therefore, these platforms present severe power and communications constrains. Taking into account these constraints and aiming to achieve a highly interoperable and versatile software component, the following design requirements where formulated for such a middleware:

- **Plug and play sensor discovery**: The middleware shall be able discover and communicate with sensors connected on-the-fly, without any prior knowledge about these sensors.

- **Standards-based configuration**: The middleware shall be able to interpret SDFs and setup an acquisition process based on the information contained in these files.
- **Cross-platform design**: The middleware shall be deployable in a maximum number of platforms, regardless of their particular hardware and software architecture.
- **Minimum resource requirements**: Due to the intrinsic constraints of some observation platforms, the usage of hardware and software resources has to be reduced as much as possible (RAM usage, bandwidth, etc.).
- **Standard compliance**: Such a middleware shall be described through SensorML files to allow systems to automatically understand its role and capabilities.

The Sensor Web Enablement Bridge (SWE Bridge) is a middleware component designed to fulfil these previously mentioned requirements. This middleware is aimed to be used as a universal driver for any sensor providing a RS232 or Ethernet interface. Its cross-platform and hardware abstraction design makes it suitable to be deployed in the majority of observation platforms, whether they are fixed or mobile.

5.2. SWE Bridge Model

The SWE Bridge has been designed following a SensorML-like style, implementing computational equivalents for the SensorML elements (*SimpleProcess*, *Parameters*, *DataRecord*, etc.) and also providing SensorML-based inheritance and configuration mechanisms (*typeOf* and *Settings*). Due to this approach, the SWE Bridge can also be modelled using the SensorML standard. The ultimate goal of this model, named SWE Bridge Model, is to provide an unambiguous description of this middleware, understandable by automated processes, providing a framework to automatically generate SDFs with minimum human intervention. This model is shown in Figure 11 and it is available online at [39].

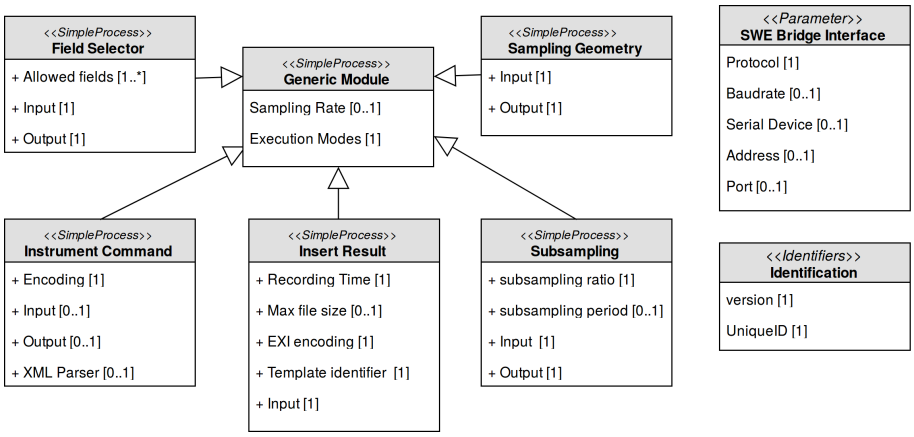


Figure 11. SWE Bridge model. All modules inherit from the generic module, where the execution options for the processes are defined. Then each module expands its definition with its particular settings. The model also contains a set of identifiers and a generic communication's interface.

The SWE Bridge model defines a generic communication interface that abstracts the physical layer, providing a protocol-agnostic communication functionality to the rest of the software. The supported protocols are serial communication, TCP and UDP.

The SWE Bridge's core is its set of modules, which are templates for observation-related, built-in processes. These processes include data retrieval, simple data manipulation and data storage among others. All these operation are described with *SimpleProcess* elements, which can be instantiated, configured and connected within a Sensor Mission as detailed in Section 4.3. Each module can be instantiated to create a process that will be executed on runtime.

The generic module is an abstract module that contains the necessary information for the coordination and operation of the resulting processes. This information is encapsulated in a set

of flags named Execution Modes, whose main function is to control the circumstances under which a particular process shall be executed.

Derived from this generic module the following built-in modules are implemented in the current version of the SWE Bridge:

- **Instrument Command**: This module provides a unified process to communicate with a sensor. Depending on the configuration, this module can be used to send any kind of commands and/or receive sensor data.
- **Field Selector**: This module allows to filter the response of a sensor, selecting the desired information and discarding the rest.
- **Subsampling**: This module allows to create subsampled data sets. It is especially useful in platforms with severe communication constraints, where a subsampled data set is transmitted in real time and a full data set is stored locally.
- **Sampling Geometry**: This module adds the platform position to a data structure, correlating sensor data with the platform's coordinates.
- **Insert Result**: This module stores the incoming data to standard O&M files, encoded in XML or EXI.

Depending on the module, it may have an input, an output or both. If a module does not have an input, it represents the beginning of the process chain (i.e., Instrument Command). On the contrary, if it has only an input and does not have output this process represents the end of a process chain (i.e., Insert Result). If a module has both input and output, the module is an intermediate process, performing data manipulation/processing. A *DataRecord* structure emulating the SWE Common Data Model standard is also used to pass data from process to process [38]. It is also possible to expand the SWE Bridge functionalities by implementing custom modules. A new module shall also inherit the parameters from the SWE Bridge generic module and follow the same data structure and input/output logic. In Section 6 different process chains for different real-world use cases are presented, including custom modules.

5.3. Implementation

In order to fulfil the cross-platform and minimum resources requirements, the SWE Bridge has been implemented using ANSI C, with special emphasis on minimizing the usage of underlying software and hardware resources. Its implementation is available online at [40]. All platform-dependent resources are abstracted using resource abstraction wrappers, which provide an unified way to access the platform's resources (see Figure 12). These wrappers are the only functions that need to be adapted when deploying the SWE Bridge in a new platform.

The SWE Bridge operation is organized in four components that are executed sequentially: the OGC PUCK Detector & Extractor, the EXI decoder, the SensorML Interpreter and the Mission Scheduler. The hardware resources needed by these components are: access to a serial and/or Ethernet communication interface, access to the observation platform's coordinates (if the platform is mobile), access to the platform's filesystem and a timer to schedule internal operations.

The execution of the SWE Bridge may vary slightly depending on whether the sensor is OGC PUCK-enabled or not, as shown in Figure 13. OGC PUCK-enabled sensors shall provide their SDF embedded within their payload memory. However, when interfacing a COTS sensor its associated SDF shall be uploaded to the platform filesystem.

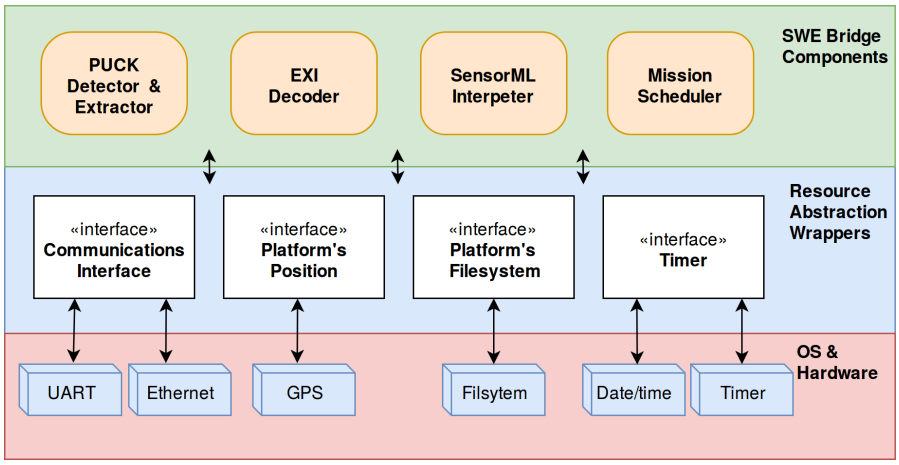


Figure 12. SWE Bridge internal architecture. The different components use the resource abstraction wrappers to hide the underlying hardware and operating system, providing a unified way of accessing platform-dependent resources.

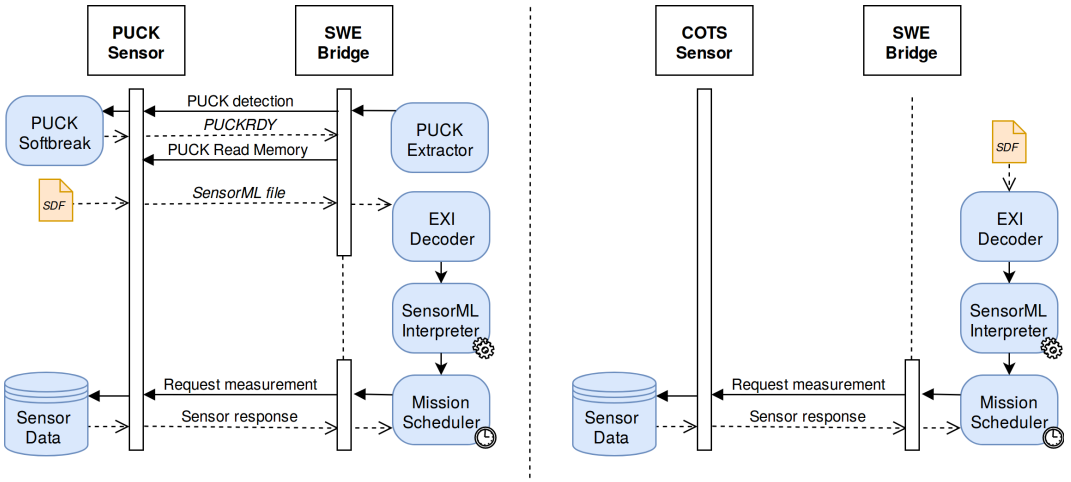


Figure 13. SWE Bridge operation. On the left the operation with a OGC PUCK-enabled sensor is shown. On the right the operation of a COTS sensor is presented, with the SDF stored locally on the platform.

The OGC PUCK Detector and Extractor detects new OGC PUCK-enabled sensors connected to the communications interface. Once a new sensor is detected, this component extracts its SDF. When interfacing a COTS sensor, a local SDF can be passed as argument to the middleware, bypassing the OGC PUCK Detector and Extractor component.

The EXI decoder, based on the EXIP framework, extracts and stores the desired information from the SDF into an intermediate structure [41]. Using a set of rules, the decoder identifies potentially useful elements to the SensorML Interpreter Service. The SensorML Interpreter service takes the extracted information and uses this data to configure the acquisition process. The first step is to configure a communication interface according to the information decoded sensor description. Afterwards this service examines the set of *SimpleProcesses* that are defined within the Sensor Mission and generates a process instance for each one of them. The generated processes are connected according to the SDF's *connections* section and finally the internal parameters of these processes are configured as specified in the *Settings* section.

Once the Auto-configuration Service has setup all necessary processes in the SWE Bridge, the Mission Scheduler is started. This is a timer-based scheduler that manages and controls the execution of the previously configured process chains.

6. Use Cases

This section illustrates different use cases where the combination of SDFs with the SWE Bridge middleware are used to successfully enable and demonstrate the plug and play integration of sensor into Sensor Web Enabled architectures.

6.1. NeXOS Project

The NeXOS project was an EU-funded project that aimed to develop cost-effective, innovative and compact multifunctional systems which can be deployed on fixed and mobile platforms [42], with special emphasis on interoperability and SWE-based architectures. Within this project the acquisition chain presented in Section 3 was arranged into the Smart Electronic Interface for Sensor Interoperability (SEISI) [43]. Different demonstration mission where performed using different sensors deployed on platforms such as gliders, underwater observatories, buoys and profilers among others. In this section the focus is put on the integration of two different NeXOS sensor developments into the SeaExplorer Glider.

Two different NeXOS-developed sensors where deployed in the SeaExplorer Glider [44], the Mini.1 and the A1 Hydrophone. The Mini.1 is an optical sensor that measures hydrocarbon concentrations in water while the A1 Hydrophone is a smart acoustical sensor with embedded real-time processing capabilities for noise measurements and mammal detection (shown in Figure 14). Both sensors implemented the OGC PUCK protocol and had their own SDF embedded within their respective payloads, describing the sensors and their mission. These files are available at [37].

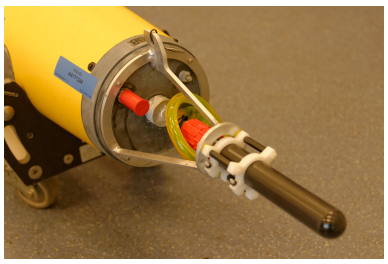


Figure 14. NeXOS A1 Hydrophone integrated on the SeaExplorer glider as payload.

Within the SeaExplorer controller, which runs an embedded Linux operating system, two instances of the SWE Bridge software were executed. These instances where in charge of retrieving and interpret the SDF and setup the data acquisition process. As the SeaExplorer glider is a mobile platform, a resource abstraction wrapper was developed in order to relate the acquired data with the vehicle position, based on socket communication between the GPS driver and the SWE Bridge. The SeaExplorer used an Iridium satellite link to communicate with the shore station. The management of this power-consuming and low-bandwidth communications in power-constrained platforms is critical. Thus, the satellite-link is controlled by the platform operators, deciding when and how the generated files will be sent to shore. The acquisition chain is showed in Figure 15.

The SWE Bridge generated two sets of data files: a subsampled data set sent in near real-time through the glider's satellite-link and a full data set, stored locally and recovered with the glider at the end of the mission. In Figure 16 the SWE Bridge mission scheduler configuration workflow is shown. The data is retrieved from the sensor by the Instrument Command process. Afterwards the Sampling Geometry process associates the latest platform position to each measurement. The data is then passed to two different branches. The first branch sends the data to an Insert Result process, which stores the full data set locally. The second branch subsamples the incoming data before passing it to another to another Insert Result process. This process stores the subsampled data set into O&M files, which is transmitted to a shore station through the glider's satellite link. A subsampled acoustic noise data set gathered by the A1 Hydrophone sent in near-real during field trials can be seen at Figure 17.

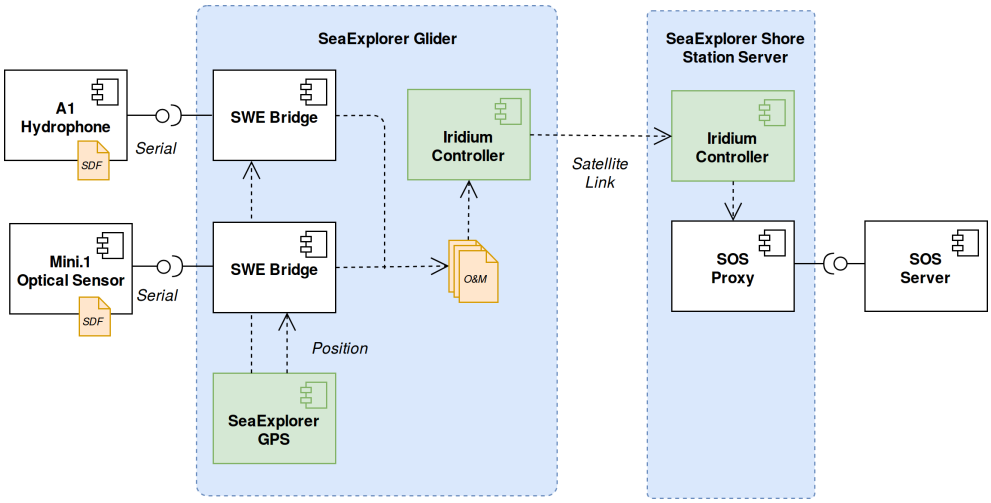


Figure 15. SeaExplorer mission acquisition chain.

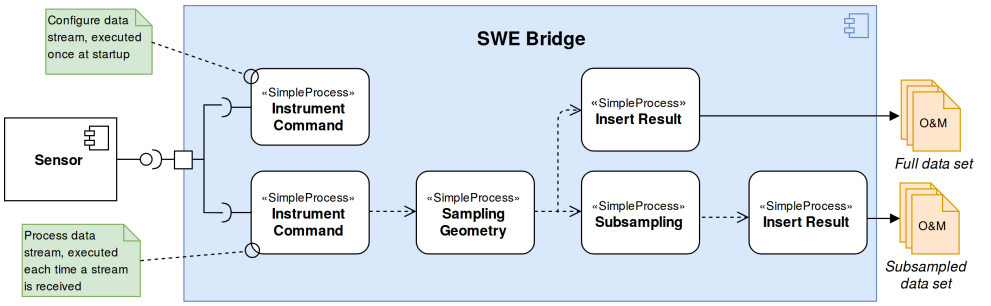


Figure 16. SWE Bridge mission scheduler configuration for the SeaExplorer NeXOS mission.

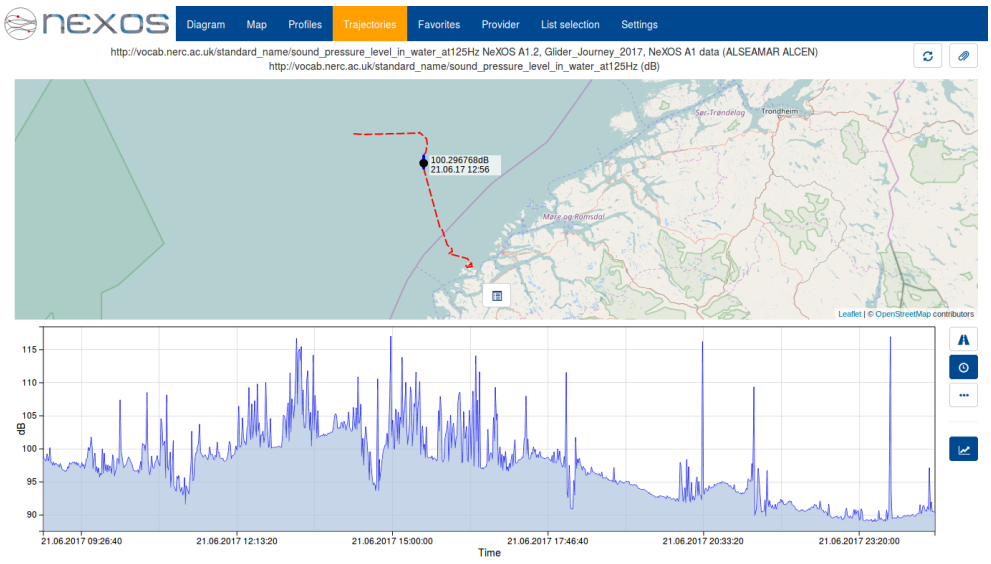


Figure 17. Sound Pressure Level (SPL) at 125 Hz octave band acquired by a SeaExplorer glider in a mission at the Norwegian coast. The incoming data from the hydrophone was subsampled and sent to shore in near real-time during deployment.

6.2. EMSODEV Project

The proposed acquisition chain has also been used within the EU-funded project EMSODEV. This project aims to develop the EMSO Generic Instrument Module (EGIM), as well as its associated

data infrastructure. The EGIM is a compact-sized observation platform designed for long-term deployments at the EMSO nodes [45]. Its main purpose is to gather extensive, multidisciplinary data sets in a standardized fashion. In order to archive and distribute the acquired data, an SWE-based infrastructure is implemented. For collecting this multidisciplinary data, the EGIM includes an instrument pack of COTS sensors, shown in Table 1. The EGIM device with its instrument pack is shown in Figure 18.

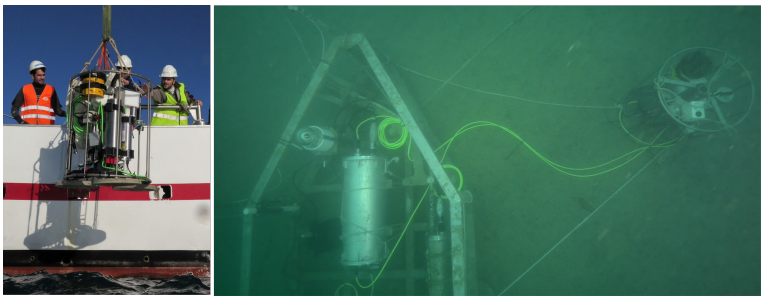


Figure 18. EMSO Generic Instrument Module (EGIM) with its instrument pack during its deployment (**left**) and EGIM deployed at the OBSEA observatory (**right**).

Table 1. EGIM Instrument Pack. The communications refers to the communications interface that the server uses to communicate to the sensors.

Sensor Type	Sensor Name	Manufacturer	Communication Link
CTD	SBE 37	SeaBird Electronics	TCP/IP
Tsunami Meter	SBE 54	SeaBird Electronics	TCP/IP
Oxygen Optode	Aanderaa 4831	Aanderaa	TCP/IP
Turbidimeter	Eco NTU	Wetlabs	TCP/IP
ADCP	Workhorse	Teledyne	TCP/IP
Hydrophone	icListen	Ocean Sonics	FTP
EGIM Internal Stuatus	EGIM	EMSODEV Consortium	UDP

The EGIM has two modes of operation: autonomous and cabled. While in autonomous mode, the EGIM is powered by internal batteries and the data coming from the instruments is logged into CSV files, retrieved after the device recovery. On the contrary, when it is operated in cabled mode, it draws power from an external source, communicating the data in real-time using an Ethernet link. The EGIM implements a serial to Ethernet converters for each sensor, which are operated externally by an acquisition server.

The EGIM device itself also sends data regarding its internal status using UDP frames. These frames contain information about input voltage, input current, remaining storage capacity, internal temperature and a leak detection alarm. Within the EMSODEV project, interoperability was also a matter of great concern. Thus, an SWE-based data acquisition process with the components presented in Section 3 was implemented [46]. Figure 19 shows the EMSODEV cyber-infrastructure in a cabled mode scenario. Each sensor is attached to the EGIM using RS232 ports, which converts this serial communication to an Ethernet link, providing a TCP/IP interface.

The acquisition server runs an instance of the SWE Bridge for each instrument deployed on the EGIM node (except the hydrophone, which records data in its internal memory). A SDF file has been written for each sensor, available at [37]. As the sensors do not implement the OGC PUCK protocol (COTS sensors), these SDFs are stored locally in the acquisition server. Another instance of the SWE Bridge is also used to decode the EGIM internal status frames, which are treated as scientific data. The SWE Bridge generates O&M files containing the acquired data. These files are passed to an SOS proxy, which injects the data to an SOS server.

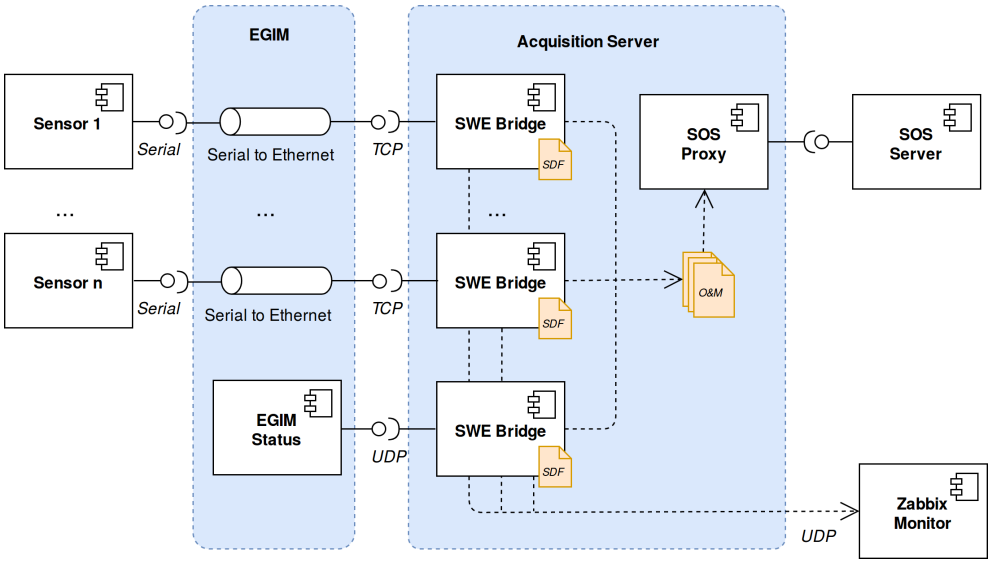


Figure 19. EGIM cyber-infrastructure in a cabled mode scenario.

Within the EMSODEV project, the use of a Zabbix monitoring system to monitor the status of the EGIM and its associated cyber-infrastructure was proposed [47]. This monitoring system does not support O&M-based transactions. Therefore, in order to fulfil the architecture requirements, the SWE Bridge functionality was expanded by implementing a specific module to send instrument data to a Zabbix monitoring server using its specific format. This module, called Send to Zabbix, sends an UDP frame to a Zabbix server for each new value arriving from the sensors.

Except the hydrophone, all the sensors including the EGIM itself are configured in streaming mode. Thus, each SDF use the same process chain for the SWE Bridge's mission scheduler, shown in Figure 20.

A first field test of the EGIM developments at the OBSEA underwater observatory was conducted from 1 December 2016 to 15 April 2017 [48]. The test showed that the cyber-infrastructure was robust and interoperable as new sensors can be easily deployed, just plugging a new sensor to an EGIM's empty slot and writing a new SDF.

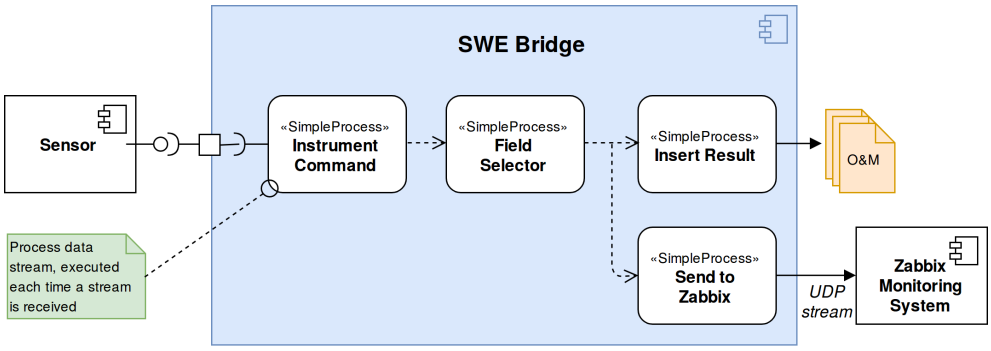


Figure 20. Mission scheduler configuration used by the SWE Bridge instances deployed at the EMSODEV acquisition server. The data coming from each sensor (all of them configured in stream mode) is acquired using an Instrument Command process, which then passes the data to a Field Selector process. This process filters the useful data and discards the undesired variables, depending on the communications protocol of each sensor. Later on, two different branches are created, one that stores the data into Observations and Measurements (O&M) files using the Insert Result process, and another one that sends the data to the Zabbix Monitoring System.

6.3. INTMARSIS Project

The INTMARSIS project aims to monitor underwater seismic activity in real-time, allowing a precise estimation of actual earthquake scales. To achieve this goal a stand-alone seismic system with real-time telemetry was designed and tested [49]. One further objective was to assess the usability of OGC compliant standardized acquisition chains in underwater seismic applications.

The INTMARSIS system, shown in Figure 21, is composed of mainly two components, an ocean bottom seismometer (OBS) and a surface buoy. The communication between the OBS and the surface buoy is performed by a stainless steel mooring line using inductive modems (SeaBird Electronics UIMM). This inductive modem provides low-bandwidth half-duplex communication through mooring lines (up to 7000 m) where regular cables are not practical [50]. The INTMARSIS system, deployed near the Catalan coast, is shown in Figure 22.

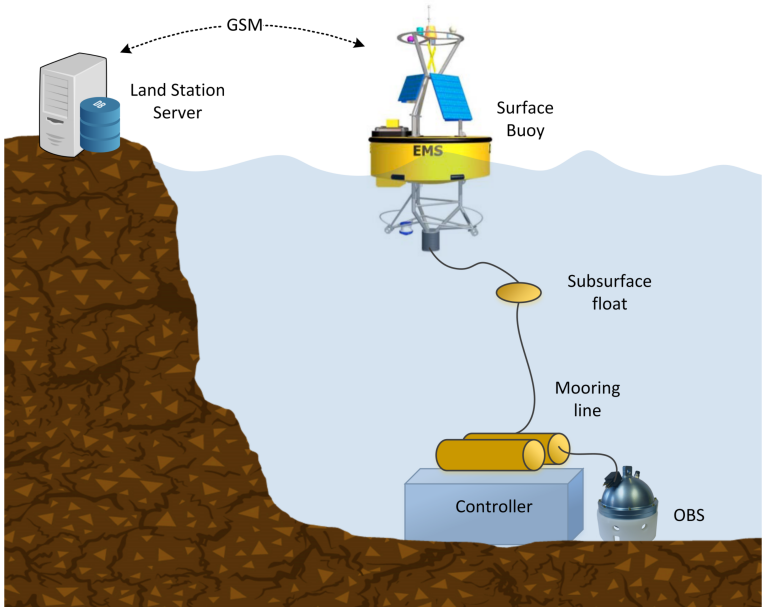


Figure 21. INTMARSIS System overview. At the seafloor the OBS (Ocean Bottom Seismometer) acquires seismic data, storing it locally. A subsampled set of data is sent through the mooring line using an inductive modem. The surface buoy receives the real-time subsampled seismic data, which is transmitted to the Land Station server using a GSM link.

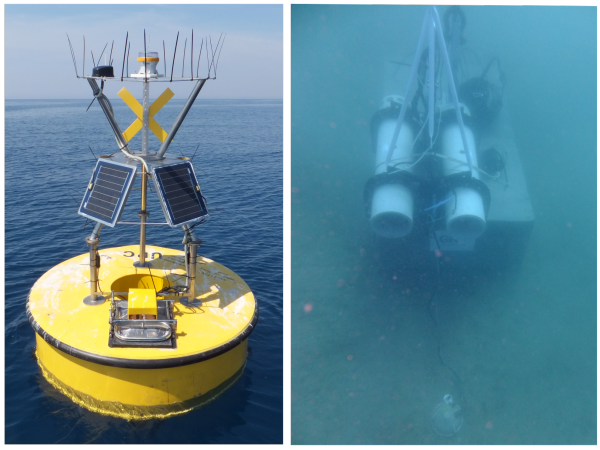


Figure 22. INTMARSIS buoy (**left**) and INTMARSIS Ocean Bottom Seismometer (**right**).

The OBS acquires 3 channels (X, Y and Z axis), taking 125 Samples per second (SPS) with a precision of 24 bits. However, due to the low bandwidth provided by the inductive modems (1200 bps), the full data set is not transmitted in real-time, but stored locally. The OBS controller generates a subsampled data set at 25 SPS which is transmitted thorough the inductive modem.

In Figure 23 the INTMARSIS acquisition chain is depicted. The subsampled data set, alongside with OBS's technical data, is sent to the buoy through the inductive link. The inductive communication and the processing of the acquired data as well as internal sensors is performed by the master controller, hosted by the surface buoy. This software component is modelled as three different Virtual Instruments (VIs): OBS technical data, buoy technical data and a peak detector.

Technical data from both OBS and buoy include internal temperature and humidity gathered by low-cost sensors integrated at the electronics board. Although this data is not scientifically relevant it may prove useful to the operators to detect hardware malfunctions and water leaks. The master controller collects the internal data and aggregates it into two different UDP streams, one for the buoy technical data and another one for the OBS technical data.

The peak detector processes the seismic data and returns the maximum absolute value during a period of time (10 s by default). Each of these VIs is interfaced by an instance of the SWE Bridge and its data sent to an SOS server at the shore station using a GSM modem. The SDF associated with each VI are available at [37].

The raw seismic data is sent in near real-time to the land station, where it is processed and stored in miniSEED files (a standard format for seismic data). This data is made publicly available through a FTP server. Although the raw seismic data is not stored by the SOS server, the peak values time series provides an indicator of the seismic activity. As the volume of data is several orders of magnitude lower than the raw seismic data, it is much easier to archive and display this data in Sensor Web environments. With this approach it is possible to discover and access data from the seismometer using Sensor Web components, and only download the seismic events instead of the whole data set.

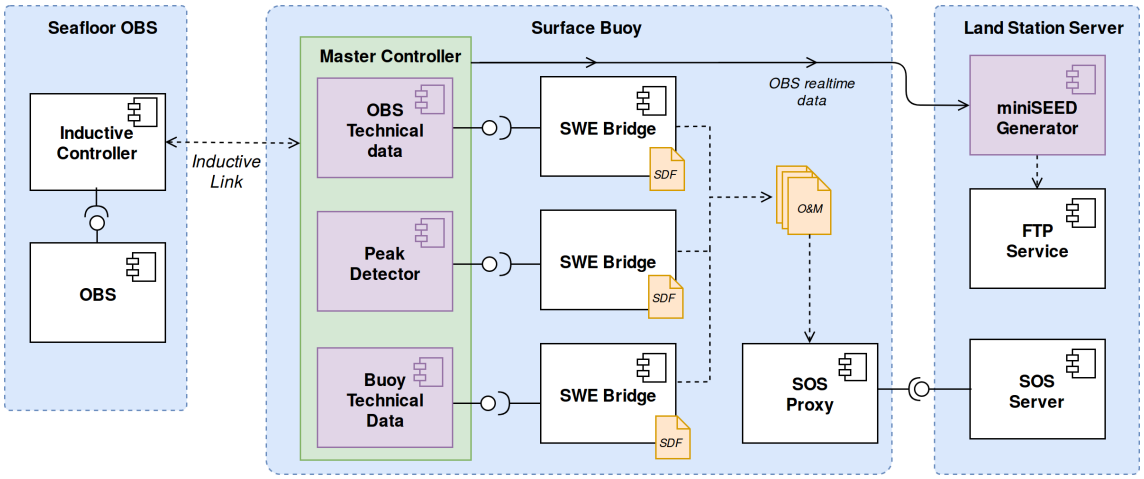


Figure 23. INTMARSIS System acquisition chain. Virtual instruments are depicted as purple components.

6.4. SWE Bridge Performance

As discussed in Section 2.1, many observation platforms present severe constraints in terms of power supply and computational resources. Although the SWE Bridge software is not performing any computationally expensive operations, it makes extensive use of dynamic memory due to the arbitrariness of SensorML documents. In this section the assessment of the SWE Bridge performance is presented when interfacing the sensors deployed in NeXOS, EMSODEV and INTMARSIS projects. In order to obtain comparable results, all tests were performed in a Raspberry Pi single board computer

(specifications shown in Table 2), acting as a host controller for the SWE Bridge. The performance was assessed using the Massif heap memory profiler [51].

Table 2. Raspberry Pi 2 specifications.

Processor Architecture	ARM 7
Processor Speed	900 MHz
Number of Cores	4
RAM memory	1 GB
Operating System	Raspbian Jessie Lite (version July 2017)

The execution of the SWE Bridge can be classified in two differentiated stages: setup (which includes the components *OGC PUCK Extractor*, *EXI Decoder* and *SDF Interpreter*) and operation (*Mission Scheduler*). During the setup process the SDF is decoded and interpreted, which produces a peak in the use of dynamic memory. Once the setup is finished, a significant amount of memory is freed and the use of dynamic memory is maintained low and constant during the operation stage. This behaviour can be observed Figure 24, where the first 30 seconds of a time-based memory profile is depicted.

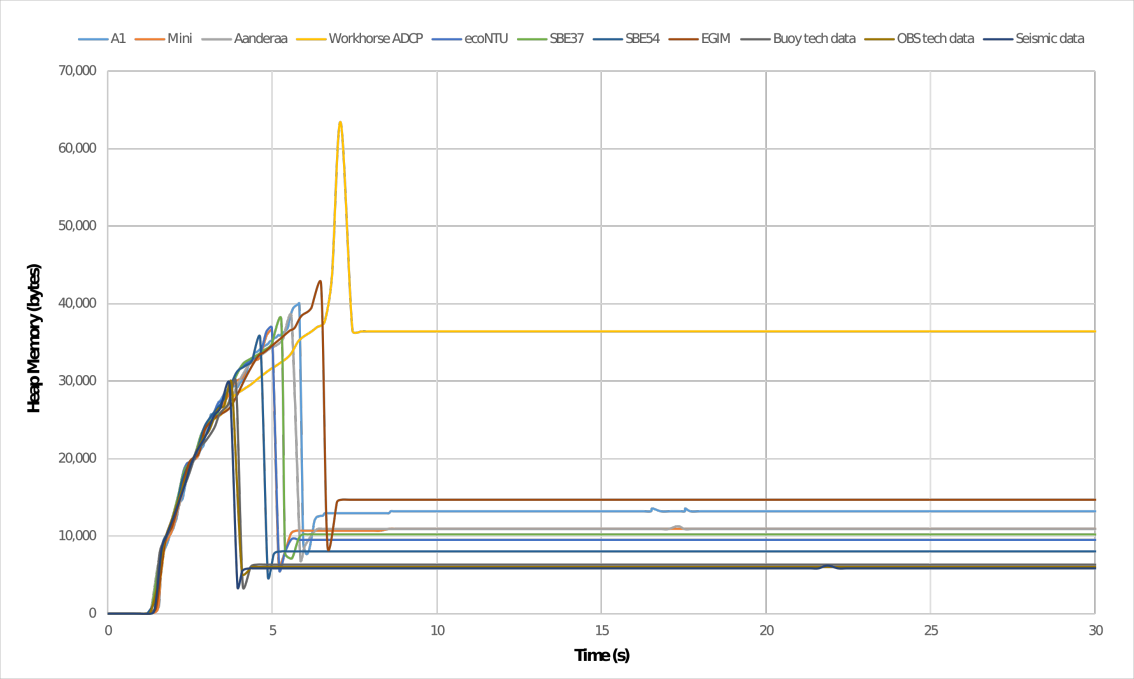


Figure 24. SWE Bridge time-based memory profile when interfacing sensors from NeXOS, EMSODEV and INTMARSIS projects. The peak usage of heap memory is registered when decoding the SDF file. Afterwards, during the operation stage, the memory consumption is kept constant.

The amount of memory used at the stationary stage is mainly dependant on the nature of the sensor response and the mission complexity. Large responses require larger buffers and each process within the SWE Bridge also increases the usage of dynamic memory. The peak usage of dynamic memory as well as the average values in stationary phase are shown in Table 3.

Table 3. SWE Bridge performance assessed when interfacing sensors from the NeXOS, EMSODEV and INTMARSIS projects. The average values of the memory consumption are calculated in the stationary phase (after the setup).

Sensor Name	Sensor Parameters				SWE Bridge Performance			
	Protocol	Stream (bytes)	Period (s)	SDF Size (bytes)	Max Heap (kBytes)	Avg Heap (kBytes)	Avg Stack (kBytes)	CPU Load (KIPS)
A1	Serial	61	1	3158	40.05	13.26	1.631	35.23
Mini.1	Serial	68	1	2451	36.63	10.96	1.568	34.50
Aanderaa 4831	TCP	80	1	3440	38.36	10.93	1.029	88.36
Workhorse	TCP	688	60	9431	63.66	36.43	0.830	78.23
Eco NTU	TCP	32	1	3290	36.93	9.53	0.892	78.26
SBE 37	TCP	72	10	3429	37.91	10.22	0.740	68.39
SBE 54	TCP	140	1	2769	35.56	8.05	0.895	87.35
EGIM	UDP	137	20	4477	42.57	14.70	0.691	68.03
Seismic Data	UDP	10	10	1728	29.96	5.84	0.692	63.61
OBS Technical	UDP	30	30	1764	29.76	6.02	0.683	63.75
Buoy Status	UDP	30	30	1764	29.96	6.32	0.685	63.89

The overall computational load, measured in kilo instructions per second (KIPS), is mainly dependant on the sensor's communications protocol, data stream period and response length. Sensors using TCP/UDP protocols increment significantly the computational load when compared to serial sensors. Moreover, sensors with short periods of data stream with large responses present the higher usage of CPU (i.e., SBE54 and Aanderaa 4831).

7. Conclusions and Future Work

In this work a framework to enable plug and play sensor integration into research data infrastructures have been proposed. It is based on the combination of different standards from the OGC's SWE framework and the W3C consortium. Using these standards a set of interoperable components have been presented to bridge between any kind of (in-situ) sensor and the Sensor Web. To ensure re-usability of the results aspects such as cross-platform support and the use of open source licenses were emphasized. These components can be easily adapted to different scenarios without any significant modification, overcoming the intrinsic constraints of ocean observation platforms.

Sensor's metadata, as well as deployment and acquisition-specific metadata are combined in a coherent format by using the concepts of SDF, providing a SensorML-based template for unambiguous sensor deployment and sensor operation description. The advantages of the combination of SDFs with the proposed acquisition chain (SWE Bridge, SOS Proxy and SOS server) has been demonstrated in three real-world scenarios. These deployments include a mobile platform with severe power and communications constraints (SeaExplorer Glider), a multidisciplinary fixed-point observation platform with a pack of commercial sensors (EGIM), and a complex seismic system (INTMARSIS system). Different sensors where integrated into observation platforms, including six COTS sensors, two newly developed OGC PUCK-enabled sensors and a complex seismic system (treated as three Virtual Instruments), each one of them with their own non-standardized proprietary protocols.

The design of the presented components and their SensorML descriptions provide the foundations for generating automated processes that can combine sensor metadata and acquisition chain metadata so that SDFs can even be created in a semi-automated manner.

Further work in this field should be focused on facilitating the application of the presented sensor acquisition chain by making the generation of SDFs easier. For example developing a user-friendly graphical user interface to generate SDFs with minimal human intervention. This tool should be able to combine the information from the sensor, the description of the acquisition chain's components and the capabilities of SOS servers in order to allow users to generate their own configuration files in an intuitive manner, hiding the specificities of the SWE standards from the end user. This would allow

users to leverage the potential of the Sensor Web without the need of in-depth knowledge of complex standards and services required.

Acknowledgments: This work has been funded by the Spanish Ministerio de Economía y Competitividad under the INTMARSIS project (contract CGL2013- 42557-R), the NeXOS Project under the European Commission 7th Framework Programme (grant agreement 614102) and the ESMODEV Project under the European Commission Research Infrastructure Programme of the H2020 (grant agreement 676555).

Author Contributions: Joaquín Del Río and Daniel M. Toma conceived the original idea. Joaquín Del Río, Daniel M. Toma and Enoc Martínez contributed to the design of the software components as well as the design of Sensor Deployment Files. Enoc Martínez implemented the software components. Simon Jirka contributed to the Sensor Web aspects. Enoc Martínez wrote the manuscript and Simon Jirka reviewed and substantially improved it.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

OGC	Open Geospatial Consortium
SWE	Sensor Web Enablement
O&M	Observations and Measurements
SOS	Sensor Observation Service
SensorML	Sensor Model Language
EXI	Efficient XML Interchange
SDF	Sensor Deployment File
COTS	Commercial off-the-shelf

References

1. Reed, C.; Botts, M.; Davidson, J.; Percivall, G. OGC Sensor Web Enablement: Overview and High Level Architecture. *GeoSens. Netw.* **2008**, *4540*, 175–190.
2. Bröring, A.; Echterhoff, J.; Jirka, S.; Simonis, I.; Everding, T.; Stasch, C.; Liang, S.; Lemmens, R. New generation Sensor Web Enablement. *Sensors* **2011**, *11*, 2652–2699.
3. Bröring, A.; Maué, P.; Janowicz, K.; Nüst, D.; Malewski, C. Semantically-enabled sensor Plug & Play for the Sensor Web. *Sensors* **2011**, *11*, 7568–7605.
4. Broring, A.; Foerster, T.; Jirka, S. Interaction patterns for bridging the gap between sensor networks and the Sensor Web. In Proceedings of the Communications Workshops 2010 8th IEEE International Conference on Pervasive Computing and Communications, Mannheim, Germany, 29 March–2 April 2010; pp. 732–737.
5. Del Rio, J.; Toma, D.M.; O'Reilly, T.C.; Broring, A.; Dana, D.R.; Bache, F.; Headley, K.L.; Manuel-Lazaro, A.; Edgington, D.R. Standards-based plug & work for instruments in ocean observing systems. *IEEE J. Ocean. Eng.* **2014**, *39*, 430–443.
6. Geraci, A.; Katki, F.; McMonegal, L.; Meyer, B.; Porteous, H. IEEE Standard Computer Dictionary. A Compilation of IEEE Standard Computer Glossaries. *IEEE Std 610* **1991**, 1, doi:10.1109/IEEESTD.1991.106963.
7. Dawes, N.; Kumar, K.A.; Michel, S.; Aberer, K.; Lehning, M. Sensor metadata management and its application in collaborative environmental research. In Proceedings of the 4th IEEE International Conference on eScience, Indianapolis, IN, USA, 10–12 December 2008; pp. 143–150.
8. Sheth, A.; Henson, C.; Sahoo, S.S. Semantic sensor web. *IEEE Internet Comput.* **2008**, *12*, 78–83.
9. Walter, K.; Nash, E. Coupling Wireless Sensor Networks and the Sensor Observation Service—Bridging the Interoperability Gap. In Proceedings of the 12th AGILE International Conference on Geographic Information Science, Hannover, Germany, 2–5 June 2009; pp. 1–9.
10. Fairgrieve, S.M.; Makuch, J.A.; Falke, S.R. PULSENet: An implementation of sensor web standards. In Proceedings of the 2009 International Symposium on Collaborative Technologies and Systems, Baltimore, MD, USA, 18–22 May 2009; pp. 64–75.
11. Geipel, J.; Jackenkroll, M.; Weis, M.; Claupein, W. A Sensor Web-Enabled Infrastructure for Precision Farming. *ISPRS Int. J. Geo-Inf.* **2015**, *4*, 385–399.

12. Sawant, S.A.; Adinarayana, J.; Durbha, S.S. KrishiSense: A semantically aware web enabled wireless sensor network system for precision agriculture applications. In Proceedings of the International Geoscience and Remote Sensing Symposium (IGARSS), Quebec City, QC, Canada, 13–18 July 2014; pp. 4090–4093.
13. Kotsev, A.; Schade, S.; Craglia, M.; Gerboles, M.; Spinelle, L.; Signorini, M. Next generation air quality platform: Openness and interoperability for the internet of things. *Sensors* **2016**, *16*, 403.
14. Kotsev, A.; Pantisano, F.; Schade, S.; Jirka, S. Architecture of a service-enabled sensing platform for the environment. *Sensors* **2015**, *15*, 4470–4495.
15. Bray, T.; Paoli, J.; Sperberg-McQueen, C.M.; Maler, E.; Yergeau, F. Extensible Markup Language (XML). *World Wide Web J.* **1997**, *2*, 27–66 .
16. Tamayo, A.; Granell, C.; Huerta, J. Using SWE standards for ubiquitous environmental sensing: A performance analysis. *Sensors* **2012**, *12*, 12026–12051.
17. Schneider, J.; Kamiya, T.; Peintner, D.; Kyusakov, R. Efficient XML Interchage (EXI) Format 1.0 (Second Edition). Available online: https://www.w3.org/TR/exi/ (accessed on 9 October 2017).
18. Song, E.Y.; Lee, K. Understanding IEEE 1451 - Networked smart transducer interface standard—What is a smart transducer? *IEEE Instrum. Meas. Mag.* **2008**, *11*, 11–17.
19. Song, E.Y.; Lee, K.B. Service-oriented sensor data interoperability for IEEE 1451 smart transducers. In Proceedings of the IEEE Intrumentation and Measurement Technology Conference (I2MTC), Singapore, 5–7 May 2009; pp. 1049–1054.
20. Liang, S.; Huang, C.-Y.; Khalafbeigi, T. *OGC SensorThings API-Part 1: Sensing*; Technical Report; Open Geospatial Consortium: Wayland, MA, USA, 2012.
21. Gigan, G.; Atkinson, I. Sensor Abstraction Layer: A unique software interface to effectively manage sensor networks. In Proceedings of the Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), Melbourne, Australia, 3–6 December 2007; pp. 479–484.
22. Broering, A.; Below, S.; Foerster, T. Declarative sensor interface descriptors for the sensor web. In Proceedings of the WebMGS 2010: 1st International Workshop on Pervasive Web Mapping, Geoprocessing and Services, Como, Italy, 26–27 August 2010; pp. 26–32.
23. Trevathan, J.; Johnstone, R.; Chiffings, T.; Atkinson, I.; Bergmann, N.; Read, W.; Theiss, S.; Myers, T.; Stevens, T. SEMAT—The next generation of inexpensive marine environmental monitoring and measurement systems. *Sensors* **2012**, *12*, 9711–9748.
24. Bröring, A.; Bache, F.; Bartoschek, T.; van Elzakker, C.P.J.M. The SID Creator: A Visual Approach for Integrating Sensors with the Sensor Web. *Lect. Notes Geoinf. Cartogr.* **2011**, 143–162.
25. Jazayeri, M.A.; Liang, S.H.L.; Huang, C.Y. Implementation and evaluation of four interoperable open standards for the internet of things. *Sensors* **2015**, *15*, 24343–24373.
26. Díaz, L.; Bröring, A.; McInerney, D.; Libertá, G.; Foerster, T. Publishing sensor observations into Geospatial Information Infrastructures: A use case in fire danger assessment. *Environ. Model. Softw.* **2013**, *48*, 65–80.
27. Bröring, A.; Janowicz, K.; Stasch, C.; Kuhn, W. Semantic challenges for sensor plug and play. In Proceedings of the International Symposium on Web and Wireless Geographical Information Systems, Maynooth, Ireland, 7–8 December 2009; Volume 5886 LNCS, pp. 72–86.
28. O'Reilly, T. *OGC® PUCK Protocol Standard Version 1.4*; Technical Report; Open Geospatial Consortium: Wayland, MA, USA, 2012.
29. Botts, M.; Robin, A. *OGC SensorML: Model and XML Encoding Standard*; Technical Report OGC 12-000; Open Geospatial Consortium: Wayland, MA, USA, 2014.
30. Jirka, S. The Marine Profiles for OGC Sensor Web Enablement Standards Team. In Proceedings of the EGU General Assembly 2016, Vienna, Austria, 23–28 April 2016; Volume 18, p. 14690.
31. Bröring, A.; Stasch, C.; Echterhoff, J. *OGC Sensor Observation Service*; Technical Report; Open Geospatial Consortium: Wayland, MA, USA, 2012.
32. Cox, S. *Geographic Information: Observations and Measurements*; Technical Report; Open Geospatial Consortium: Wayland, MA, USA, 2013.
33. SOS Proxy (java). Available online: https://bitbucket.org/swebridgedevelopment/sos-proxy-java (accessed on 25 October 2017).
34. SOS Proxy (Bash). Available online: https://bitbucket.org/swebridgedevelopment/sos_proxy (accessed on 24 October 2017).

35. 52° North Sensor Observation Service (SOS). Available online: https://github.com/52North/SOS (accessed on 24 October 2017).
36. Helgoland. Available online: https://github.com/52North/helgoland (accessed on 25 October 2017).
37. Sensor Deployment File Repository. Available online: https://bitbucket.org/swebridgedevelopment/sdfs (accessed on 2 December 2017).
38. Robin, A. *OGC SWE Common Data Model Encoding Standard*; Technical Report; Open Geospatial Consortium: Wayland, MA, USA, 2011.
39. SWE Bridge Model. Available online: https://www.upc.edu/cdsarti/OBSEA/SWE/files/swe_bridge/model/swe_bridge_model.xml (accessed on 6 November 2017).
40. SWE Bridge. Available online: https://bitbucket.org/swebridgedevelopment/swebridge (accessed on 24 October 2017).
41. Kyusakov, R.; Pereira, P.P.; Eliasson, J.; Delsing, J. EXIP: A Framework for Embedded Web Development. *ACM Trans. Web* **2014**, *8*, 23:1–23:29.
42. Delory, E.; Castro, A.; Waldmann, C.; Rolin, J.F.; Woerther, P.; Gille, J.; Del Rio, J.; Zielinski, O.; Golmen, L.; Hareide, N.R.; et al. Objectives of the NeXOS project in developing next generation ocean sensor systems for a more cost-efficient assessment of ocean waters and ecosystems, and fisheries management. In Proceedings of the MTS/IEEE OCEANS 2014: Oceanic Engineering Society (OES) and Marine Technology Society, Taipei, Taiwan, 7–10 April 2014.
43. Toma, D.M.; Del Rio, J.; Jirka, S.; Delory, E.; Pearlman, J.; Waldmann, C. NeXOS smart electronic interface for sensor interoperability. In Proceedings of the MTS/IEEE OCEANS 2015: Discovering Sustainable Ocean Energy for a New World, Genova, Italy, 18–21 May 2015.
44. Claustre, H.; Beguery, L. SeaExplorer Glider Breaks Two World Records. *Sea Technol.* **2014**, *55*, 19–21.
45. Favali, P.; Dañobeitia, J.; Beranzoli, L.; Rolin, J.F.; Lykousis, V.; Ruhl, H.A.; Paul, G.; Piera, J.; Huber, R.; del Río, J.; et al. European Multidisciplinary and Water-Column Observa tory—European Research Infrastructure Consortium (EMSO ERIC): Challenges and opportunities for Strategic European Marine Sciences. In Proceedings of the 7th International Workshop on Marine Technology, Barcelona, Spain, 26–28 October 2016; pp. 100–103.
46. Toma, D.M.; Del Rio, J.; Cadena, J.; Bghiel, I.; Martínez, E.; Nogueras, M.; Garcia, Ó.; Dañobeitia, J.; Sorribas, J.; Casas, R.; et al. OGC SWE-based data acquisition system development for EGIM on EMSODEV EU project. In Proceedings of the Geospatial Sensor Webs Conference, Münster, Germany, 29–31 August 2016; Volume 1762, pp. 1–5.
47. Zabbix. Available online: https://www.zabbix.com (accessed on 15 November 2017).
48. Aguzzi, J.; Mànuel, A.; Condal, F.; Guillén, J.; Nogueras, M.; del Rio, J.; Costa, C.; Menesatti, P.; Puig, P.; Sardà, F.; et al. The new seafloor observatory (OBSEA) for remote and long-term coastal ecosystem monitoring. *Sensors* **2011**, *11*, 5850–5872.
49. Toma, D.M.; Artero, C.; Del Río, J.; Trullols, E.; Roset, X. Near Real-Time Determination of Earthquake Source Parameters from the Coastal Ocean. In Proceedings of the 7th International Workshop on Marine Technology, Barcelona, Spain, 26–028 October 2016.
50. Real-Time Oceanography with Inductive Moorings and Inductive Modem Module. Available online: http://www.seabird.com/sites/default/files/documents/AppNote92Oct16.pdf (accessed on 6 October 2017).
51. Massif: A Heap Profiler. Available online: http://valgrind.org/docs/manual/ms-manual.html (accessed on 28 November 2017).

A.2 Article 2

Metadata-driven Universal Real-time Ocean Sound Measurement Architecture

---

Journal: *IEEE Access*
Publisher: IEEE
Date of Publication: 23 February 2021
Impact Factor: 3.75
JCR Rank: Q1 Electronics
DOI: 10.1109/ACCESS.2021.3058744

IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Metadata-driven Universal Real-time Ocean Sound Measurement Architecture

ENOC MARTÍNEZ[1], ALBERT GARCÍA-BENADÍ[1], DANIEL M. TOMA[1], ERIC DELORY[2], (Senior Member, IEEE), SPARTACUS GOMÁRIZ[1], (Member, IEEE), JOAQUÍN DEL RÍO [1] (Senior Member, IEEE)

[1]Universitat Politècnica de Catalunya, Electronics Department, SARTI-MAR Research Group, Vilanova i la Geltrú, 08800, Spain
[2]Oceanic Platform of the Canary Islands, Ctra de Taliarte s/n, Telde, 35200, Spain

Corresponding author: Enoc Martínez (e-mail: enoc.martinez@upc.edu).

This work has been funded by the Spanish government project RESBIO (grant agreement TEC2017-87861-R), the EU projects ENVRI-FAIR (grant agreement 824068), the JONAS INTERREG Atlantic Area-funded project (EAPA_52/2018), partly by the Generalitat de Catalunya SARTI-MAR project (grant agreement 2017 SGR 371) and partly by the FPI-UPC_2015 scholarship program.

ABSTRACT

Underwater sound in the oceans has been significantly rising in the past decades due to an increase in human activities, adversely affecting the marine environment. In order to assess and limit the impact of underwater noise, the European Commission's Marine Strategy Framework Directive (MSFD) included the long-term monitoring of low-frequency underwater sound as a relevant indicator to achieve a good environmental status. There is a wide range of commercial hydrophones and observing platforms able to perform such measurements. However, heterogeneity and lack of standardization in both hydrophones and observing platforms makes the integration and data management tasks time-consuming and error-prone. Moreover, their power and communications constraints need to be addressed to make them suitable for long-term ocean sound monitoring. Measured underwater sound levels are challenging to compare because different measurement methodologies are used, leading to a risk of misunderstandings and data misinterpretation. Furthermore, the exact methodology applied is not always public or accessible, significantly reducing ocean sound data re-usability. Within this work, a universal architecture for ocean sound measurement is presented, addressing hydrophone integration, real-time *in situ* processing and data management challenges. Emphasis is placed on generic and re-usable components, so it can be seamlessly replicated and deployed in new scenarios regardless of the underlying hardware and software constraints (hydrophone model, observing platform, operating system, etc.). Within the proposed architecture, a generic implementation of an underwater sound algorithm based on underwater noise measurement best practices is provided. Standardized and coherent metadata with emphasis on strong semantics is discussed, providing the building blocks for FAIR (Findable, Accessible, Interoperable, Reusable) ocean sound data management.

INDEX TERMS Ocean Sound, Underwater Acoustics, Sensor Web Enablement, Interoperability, Real-time Systems, Data Acquisition

I. INTRODUCTION

THERE has always been underwater background sound in the oceans due to natural factors. However, human activities such as shipping, construction, sonar and seismic exploration have been raising the underwater ambient noise to unprecedented levels in the past decades [1]. Ocean sound is an important environmental factor for many species, especially to those using underwater sound for localization and communication [2]. Thus, the introduction of energy into the marine habitat in the form of acoustic noise needs to be properly monitored and studied to minimize its harmful impact on the ecosystem.

In order to achieve a good environmental status in European waters, the European Parliament approved the Marine Strategy Framework Directive (MSFD) which aims to protect the marine environment and ecosystem. This directive includes a set of indicators to measure the status of European waters, including maximum ocean underwater sound levels considered as acceptable (MSFD indicator 11.2.1). This indicator requires the long-term measurement of Sound Pressure

Level (SPL) at the 1/3 octave bands centered at 63 and 125 Hz. It has also been suggested to extend the monitored bands from 10 Hz up to 20 kHz in order to achieve a more accurate assessment of underwater noise [3].

The term "underwater noise" usually refers to the anthropogenic sound that has the potential to cause negative impacts on marine life, while "underwater ambient noise" usually refers to the background sound with no distinguishable sources [4]. Within this work, the generic term "ocean sound" is used to refer to the overall underwater sound level, regardless of its source or its impact on marine life.

A. OCEAN SOUND MEASUREMENT SYSTEMS

As concern about underwater noise increased, numerous studies have been presented analyzing ocean sound [5], [6], [7], [8]. Commercial off-the-shelf hydrophones used in ocean sound monitoring usually provide raw acoustic data, whether as acoustic recordings or in streaming mode. Thus, acoustic data has to be post-processed to obtain meaningful underwater sound levels. Due to the high sampling rate used by acquisition or recording systems (usually from tens to hundreds of kHz), streaming raw acoustic data from an observing platform to a shore station is only possible with broadband communications. Cabled observatories equipped with hydrophones are an excellent observing platform for long-term ocean sound monitoring, since they do not have constraints regarding bandwidth and power [9], [10]. However, these infrastructures are scarce and costly to maintain.

Some surface buoys equipped with broadband radio communications are also capable of streaming acoustic data in real-time to shore station for real-time processing [11]. However, the autonomy of these systems is greatly constrained by their power availability. These buoys also need to be located close to shore, since broadband radio links have limited coverage range, satellite transmission being currently cost-prohibitive.

Moored autonomous recorders composed of one or several hydrophones and a recording unit have also been used in large-scale deployments for underwater noise assessment [6], [8], [12]. The autonomy of such devices is mainly reduced by two factors: power and storage capacity. The storage of acoustic data may require gigabytes per day depending on the sampling rate. Usually data acquired by these devices are only available after recovery, so they do not provide real-time capability.

Underwater gliders have also been used to sense the underwater soundscape [13], [14]. Although underwater gliders have intermittent communication link to shore stations during surface time, their satellite communications have very limited bandwidth, not suitable for raw acoustic data transmission.

Telemetry is usually one of the more power-demanding components of an autonomous system such as underwater gliders. Thus, reducing the data transmission (and its associated power consumption) is vital to extend their autonomy. If the acoustic data could be processed *in situ*, the amount of information to be transmitted would decrease by several orders of magnitude, allowing real-time measurements from autonomous platforms with satellite telemetry, such as gliders, profilers, moored buoys, etc.

However, processing acoustic data in real-time is not a trivial task due to the required computational power and intrinsic complexity of acoustic signals. Although there are hydrophone prototypes with embedded underwater sound level algorithms, they are not yet widely used and their embedded algorithms still have room for improvement [15].

B. STANDARDIZATION AND INTEROPERABILITY

Scientific instruments, including hydrophones, tend to use proprietary and non-standardized protocols. Thus, in order to integrate these instruments to observing platforms, *ad hoc* drivers are usually developed. Furthermore, observing platforms have a broad range of software and hardware architectures (operating systems, communications link, computational/power constraints), so these *ad hoc* drivers need to be developed for each instrument-platform combination. This lack of code re-usability is costly, time consuming and requires in-depth knowledge of both instrument and observing platform functionalities [16].

There have been several attempts to facilitate instrument integration by standardizing instruments protocols, such as IEEE 1451 [17]. However, instrument manufacturers did not embrace this approach and still use proprietary interfaces. Other approaches try to describe instrument interfaces using machine-understandable descriptions [16]. Using this approach, interoperability can be achieved even if manufacturers do not adopt a common standard.

Beyond sensor integration, data management and interoperability also prove challenging for the ocean observing community. Data and metadata formats and procedures usually vary significantly across domains and institutions, leading to information silos and preventing the data to be effectively shared across different scientific communities. This lack of standardization has been the driving force of the Sensor Web Enablement (SWE) set of standards, supported by the Open Geospatial Consortium (OGC) [18]. SWE provides a standard framework for data and metadata ingestion, archival and retrieval with a strong focus on robust semantics and machine-to-machine interactions. However, the integration of sensor data to the SWE framework is not trivial and several approaches have been proposed [19], [20], [21].

Within this paper, a universal architecture for *in situ*, real-time ocean sound monitoring is proposed, compliant with the needs of the ocean observing community (i.e. MSFD indicators) and following best practices on underwater sound measurement methodologies [22], [23]. This architecture addresses interoperability issues at both sensor integration and data management levels by proposing a solution based on the SWE framework. Emphasis is placed on generic and re-usable components, so the proposed architecture can be replicated and deployed in new scenarios regardless of the underlying hardware and software constraints (hydrophone model, observing platform, operating system, etc.). This

work relies on the Sensor Model Language (SensorML) and Observations and Measurements (O&M) for coherent and semantically tied data and metadata [24], [25]. The ultimate goal of the proposed architecture is to provide the building blocks for ocean sound data management following the FAIR principles (Findable, Accessible, Interoperable and Reusable) [26] [27].

The paper is structured as follows. In section II an ocean sound measurement architecture is presented. Section III discusses an underwater sound level algorithm and its implementation. In section IV the proposed interoperability and metadata solution is discussed. In section V three use cases covering different scenarios are presented. The paper closes with conclusions and an outlook to future work.

II. UNIVERSAL OCEAN SOUND MEASUREMENT ARCHITECTURE

Metadata plays an important role during the data life-cycle, providing vital context to measurements: what was measured, where it was measured, who led the deployment, etc. However, metadata is usually compiled after data is acquired and targets only the measurements themselves, providing little information of the overall acquisition chain.

Within this work, a step forward is taken, using the metadata not only to provide contextual information, but also to configure a generic data acquisition chain. In other words, the data acquisition, processing and storage processes are driven by the sensor's metadata. So, metadata is not added to the acquired data once gathered, but prepared beforehand and controls the acquisition process. Using this approach, metadata may not only reflect what has been measured, but also unambiguously define the whole acquisition chain, including sensor setup, signal processing, formatting, etc. Thus, the acquisition chain in a deployment can be easily replicated based on its metadata.

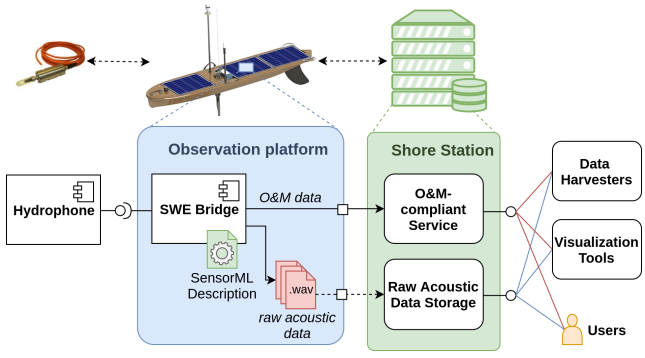


FIGURE 1. Proposed architecture for ocean sound monitoring. Hydrophone raw data is processed *in situ* by the SWE Bridge according to its SensorML metadata file. Then, two streams of data are sent to the shore station: underwater sound levels (sent in real-time) and acoustic recordings (real-time or delayed, depending on the telemetry used). At the shore station data is made available to users, data harvesters, visualization tools and others using standardized interfaces.

Ocean sound is a non-trivial variable which requires complex signal processing. Slight differences in the acquisition and signal processing may lead to significant differences in the result. Thus, it is important to state the exact method applied. Within this work the concept of a metadata-driven measurement architecture is applied to ocean sound. Using this approach, metadata contains unambiguous information about the whole acquisition chain, from the instrument's low-level configuration to the details of signal processing.

The dataflow of this metadata-driven architecture, depicted in Fig. 1, starts with a generic hydrophone acquiring raw acoustic data. Since all its low-level specifications will be abstracted, almost any hydrophone may be used. The next components in the architecture are the SWE Bridge and the hydrophone's metadata description in SensorML format [24]. The SWE Bridge is an open source, standards-based universal driver [19]. One of its key features is its ability to interface scientific instruments without any previous knowledge using a SensorML description, regardless of its vendor-specific protocols. It can manage sensor communications in almost any format, ranging from plain ASCII communications through serial port to Ethernet high frequency binary streams (e.g. hydroacoustic data).

The SWE Bridge has been designed bearing in mind the heterogeneity of observing platforms and scientific instruments. Its re-usable and generic design facilitates the deployment in any platform. Considering power and communications constraints of observing platforms, stress has been put on an efficient and modular implementation using very limited computational resources and optimized telemetry. Its modularity and versatility have been proved in numerous deployments using various observing platforms, such as unmanned surface vehicles, underwater gliders, cabled observatories and autonomous buoys, among others.

The SWE Bridge includes an acoustics module that provides real-time underwater sound levels following the recommendations of the MSFD Task Group 11 on underwater noise [3]. Sound Pressure Level (SPL), Root Mean Squared pressure (RMS) and Sound Exposure Level (SEL) at 1/3 octave bands are calculated. Moreover, the acoustics module is not limited to the MSFD requirements and can be easily configured to monitor any band within the hydrophone's frequency range. The internal processing is discussed in detail in section III.

The metadata encoded in the SensorML description contains extensive information about the hydrophone's characteristics, deployment details, communication protocol, setup routines and more. The SWE Bridge is also configured through this metadata file, selecting different processing and formatting options. Thus, the SensorML file does not only contain hydrophone's metadata, it contains all the required information to replicate the deployment: from sensor configuration to underwater sound level algorithm details.

Since the SWE Bridge can be deployed within any observation platform, the hydrophone's data stream is processed *in situ* and in real-time. Ocean sound levels are usually required for time windows of tens of seconds, so the processed data stream requires several orders of magnitude less storage

and bandwidth than the raw data. The main advantage of processing the data on-board of the observation platform is that processed data may be transmitted in real-time, even if the bandwidth is limited (e.g. satellite communications). On the contrary, raw acoustic data (acoustic recordings) has be stored internally until recovery (acoustic recorder, underwater glider, etc.), unless broadband communications are available (e.g. cabled observatory).

The SWE Bridge generates processed data following the O&M standard. Thus, underwater sound levels may be injected in real-time into an O&M-compliant service such as Sensor Observation Service (SOS) or SensorThings [28], [29]. Although the preferred format for underwater sound levels is the metadata-enriched O&M format, CSV (comma separated values) output may also be provided by the SWE Bridge. Due to its large volume and their specific nature, acoustic recordings can be archived in a generic data storage service such as file transfer protocol (FTP) or ERDDAP [30].

Once the data are injected to a cyber-infrastructure, data and metadata are managed and disseminated following the FAIR principles. Further services can query and access both data and metadata, including users, data visualization tools and data harvesters (e.g. data assembly centers and portals). At this stage, the critical importance of metadata arises, since users and machines interacting with the cyber-infrastructure may not know beforehand the acquisition and processing details. Robust semantics and standardized metadata formats facilitate the interpretation and contextualization of the acquired data to both human and machines.

III. UNDERWATER SOUND LEVEL ALGORITHM

Measured Underwater sound levels are sometimes difficult to compare because different measurement methodologies or acoustic metrics are used, leading to a risk of misunderstandings between scientists from different disciplines [23]. The goal of the presented architecture is to obtain underwater sound measurements using appropriate metrics, following best practices on the field and community-accepted procedures [3], [22], [23].

This work focuses on providing an implementation of ocean sound algorithms that can be seamlessly integrated into resource-constrained platforms to provide real-time, *in situ* measurements. One of the critical aspects for these platforms is the computational cost required to apply underwater sound level algorithms in real-time, while maintaining a reasonable frequency resolution (Δf). In order to select the algorithm's implementation a computational cost analysis of different techniques was performed, focusing on execution time and memory usage (see appendix A).

Ocean sound is usually measured using the Sound Pressure Levels (SPL) over a time window T at different 1/3 octave band frequencies [23]. Both the time window and the frequency bands may be adjusted by the operator depending on the deployment. Finally, the algorithm implementation within the SWE Bridge is discussed.

A. SIGNAL CONDITIONING STAGE CHARACTERIZATION

In order to calculate SPL levels, the first step is to obtain the pressure signal. Acquisition systems or digital hydrophones usually do not provide the pressure signal, but the dimensionless output from the analog-to-digital converter (ADC). Thus, the signal conditioning stage has to be characterized. The typical signal conditioning stage of a hydrophone, depicted in Fig. 2, contains a transducer, a pre-amplifier and an ADC converter. The following signal conditioning properties need to be known:

- S : transducer sensitivity (dB re 1 V/µPa)
- G: pre-amplifier gain (dB)
- V_{ref}: ADC's reference voltage (V)
- M: ADC's resolution in bits (dimensionless)
- f_s: Sample Rate (Hz)

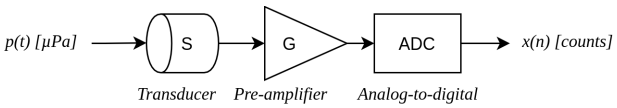


FIGURE 2. Generic hydrophone acquisition chain. Underwater pressure is converted to an analog voltage by the transducer. Its output is amplified by the preamplifier and finally it is converted to a discrete signal using an analog-to-digital converter.

Once these parameters are known, i.e. encoded in the SensorML file, the instantaneous pressure (also named pressure signal) can be calculated from the raw dimensionless samples with (1). Note that S and G have been converted from dB to linear, S_{lin} and G_{lin}.

$$p(n) = \frac{x(n)}{S_{lin} G_{lin} \dfrac{1}{LSB}} [\mu Pa] \tag{1}$$

Where $x(n)$ is the raw digital counts provided by the ADC and LSB^{-1} (least significant bit) is the ADC conversion coefficient in $counts/V$.

The LSB can be calculated from the ADC's specifications using (2):

$$LSB = \frac{V_{fs}}{2^M} = \frac{2V_{ref}}{2^M} \; [V/counts] \tag{2}$$

Where M is the ADC's number of bits, V_{fs} is the ADC's full-scale voltage range and V_{ref} is ADC's reference voltage. In (2), it has been assumed that the hydrophone's ADC has a symmetric reference voltage ($V_{ref+} = -V_{ref-}$).

Some hydrophones already provide the total sensitivity S_{total} of the hydrophone considering the amplifier gain and the ADC's conversion coefficient. In this case the sensitivity is given in dB re count/µPa and the raw dimensionless samples may be directly converted to pressure using (3) (note that total sensitivity has been converted to linear units, S_{total}).

$$p(n) = \frac{x(n)}{S_{total}} [\mu Pa] \tag{3}$$

Once the input signal has been converted from dimensionless samples to a discrete pressure signal $p(n)$ in µPa, the Sound Pressure Level (SPL) over a time interval T can be calculated applying (4) [23].

$$SPL_N = 10\log_{10}\left(\frac{1}{N}\sum_N \frac{p(n)^2}{p_0^2}\right)[dB\ re\ \mu Pa] \quad (4)$$

Where N is the number of samples within the time window ($N = T \cdot f_s$) in the pressure signal's segment and the reference pressure in seawater is p_0 = 1 µPa.

In (4) the whole bandwidth of the pressure signal $p(n)$ is considered. However, usually it is not desirable to provide the overall SPL value since it is highly dependent on the hydrophone's bandwidth. In order to avoid the dependency on the sampling rate, SPL values are calculated over a frequency band, generally 1/3 octave bands as required by the MSFD indicators.

B. 1/3 OCTAVE BAND FREQUENCIES

The calculation of SPL values over a specific frequency band can be achieved by different means. Within this work the Fast Fourier Transform (FFT) is used to estimate the SPL over a set of frequency bands. Other methods such as a filter bank and Goertzel's algorithm were analyzed to compute SPL levels, but the FFT proved to be the most robust and computationally efficient (see annex A).

In order to calculate the SPL over a frequency band, the first step is to estimate the power spectral density (PSD) of the pressure signal by means of its periodogram:

$$P_{xx}(f) = \frac{1}{N}\left|P(f)\right|^2 [\mu Pa^2/Hz] \quad (5)$$

Where P_{xx} is the periodogram over a time interval $T = N/fs$, $P(f)$ is the Discrete Fourier Transform (DFT) of the discrete pressure signal $p(n)$ and f is the normalized frequency [31].

The DFT assumes that a signal is stationary and periodic, which is not the case in acoustics signals. This causes the DFT to "see" discontinuities around the edges of the time window (segment of length N), which leads to spectral leakage. The spectral leakage is the spread of power from one DFT bin to its adjacent bins. In order to reduce the spectral leakage, it is a common practice to apply a smoothing window function to the signal before calculating its DFT, as shown in (6) [32]:

$$p_w(n) = w(n) \cdot p(n) \quad (6)$$

Where $w(n)$ is a window function of length N samples and $p_w(n)$ is the resulting windowed pressure signal. Applying a window reduces the spectral leakage, but it has severe implications on the resulting spectrum [32]. The resulting $p_w(n)$ signal amplitude is reduced around the edges of the time window, minimizing the discontinuities. However, since the amplitude is reduced, its overall energy diminishes. In order to correct this loss of amplitude the result of the DFT has to be scaled with the coherent gain (CG). This factor can be calculated as the sum of the window components (W) normalized by the number of samples (N), as showed in (7) and (8).

$$W = \sum_N w(n) \quad (7)$$

$$CG = \frac{W}{N} \quad (8)$$

From the spectral point of view, the amount of energy within each DFT bin is also modified by the window function. Each DFT bin can be understood as a very narrow band-pass filter with a Δf bandwidth. However, when a window is applied, the bandwidth of this hypothetical filter is slightly increased due to the aperture of the window's spectral response main lobe [33].To correct the extra power contribution in each DFT bin due to the bandwidth increment, the normalized equivalent noise bandwidth ($nenbw$) correction factor is used. The $nenbw$ factor can be calculated using (9) [33].

$$nenbw = N\frac{\sum_N w(n)^2}{W^2} \quad (9)$$

In order to compensate the mentioned side-effects of windowing, the periodogram of a windowed signal (also known as modified periodogram) can be calculated using (10) [31].

$$P'_{xx}(f) = \frac{1}{N \cdot nenbw}\left|\frac{P_w(f)}{CG}\right|^2 = \frac{|P_w(f)|^2}{\sum_N w(n)^2} \quad (10)$$

Being $P_w(f)$ the DFT transform of the windowed pressure signal $p_w(n)$. Note that $nenbw$ and CG corrections assume that the input signal is stationary and has a flat spectral response, i.e. white Gaussian noise. In real-world acoustic signals this is rarely the case, so a small error due to windowing side-effects is expected.

One of the weak points of the periodogram as a PSD estimator is its high variance. In order to reduce the variance of each estimate, several periodograms can be averaged using the Bartlett's or Welch's methods. However, when averaging periodograms there is a trade-off between the frequency resolution Δf and the variance reduction [34]. On the other hand, SPLs are usually calculated over large periods of time (usually tens of seconds), so averaging can help to reduce the computational costs in terms of memory and number of operations [35].

The Bartlett's method slices the signal with M samples into K non-overlapping, sequential segments ($M = K \cdot N$). Then it calculates the periodogram for each segment and averages the results (Fig. 3, top) [34]. This approach does not require CG nor $nenbw$ corrections, but the effect of the spectral leakage is greater.

The Welch method is similar, but the slices are windowed and overlapped (Fig. 3, bottom) [34], [35]. This method

has significantly less spectral leakage, but uses the CG and $nenbw$ corrections, which may also induce errors due to the non-stationary and non-gaussian nature of real-world acoustic signals. Additionally, an increase of computational cost due to the FFT overlapping is expected.

Then, the power spectral density can be estimated using the Welch's method or the Bartlett's method by averaging K periodograms:

$$\overline{P}_{xx}(f) = \frac{1}{K} \sum_{i=0}^{i=K-1} P'_{xx_i}(f) \tag{11}$$

Where $\overline{P}_{xx}(f)$ is the power spectral density estimation, K is the number of periodograms being averaged and $P_{xx_i}(f)$ is the periodogram of the ith data segment. The number of segments to be averaged depends on the length of each data segment N and on the overlapping [35].

Once the power spectral density has been estimated, the SPL value within a frequency band over a time interval $T' = M/f_s$ can be calculated with (12).

$$SPL_{f,N} = 10\log_{10}\left(\frac{2}{N}\sum_{f_l}^{f_h}\overline{P}_{xx}(f)\right)\,[dB\ re\ \mu Pa] \tag{12}$$

Being f_l and f_h the low / high limit frequencies of the desired third-octave band and N is the number of samples within each FFT segment. Since the PSD of a real signal is symmetric, the negative frequencies are redundant and can be omitted [33]. However, to consider their energy contribution, a factor of 2 has to be applied to the sum of the positive frequency bins. Note that the result of (12) depends on multiple parameters, including the total number of samples M, the number of samples in each data segment N, the overlapping between data segments and the window function.

In (12) several adjacent bins are added to approximate a third-octave band. However, there is only a finite number of frequency bins and f_l and f_h may not coincide with them. For instance, the pressure signal's power spectrum calculated over a time period of 1 second has a bin width of 1 Hz. If the SPL in the third-octave band centered at 63 Hz is calculated, the edge frequencies are 56.6 Hz and 70.8 Hz, which are not aligned with the frequency bins. In order to correct this misalignment a bandwidth correction needs to be applied, as shown in (13) [22].

$$SPL'_{f,N} = SPL_{f,N} - 10\log_{10}\frac{BW_{real}}{BW_{ideal}} \tag{13}$$

Where $SPL'_{f,N}$ is the sound pressure level value with the bandwidth correction, BW_{ideal} is the ideal bandwidth of the third octave band (f_h - f_l) and BW_{real} is the real bandwidth summed in (12).

C. ALGORITHM IMPLEMENTATION

The algorithm previously presented has been integrated within the SWE Bridge software to provide a universal ocean sound monitoring tool.

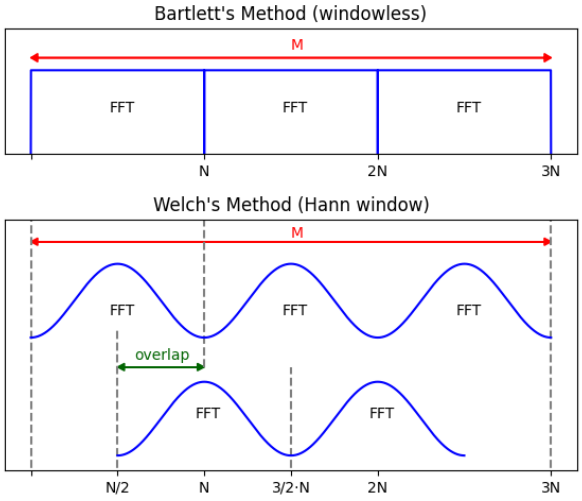


FIGURE 3. Welch's and Bartlett's methods for power spectral density estimation. The Bartlett's method uses sequential, non-overlapped, windowless data segments. The Welch method uses overlapped windowed data segments, in this example it uses a Hann window and a 50% overlap.

Usually SPL values are computed in time windows of tens of seconds, e.g. 20 seconds [3]. Within the SWE Bridge this can be adjusted by the user to match different deployment scenarios.

There is strong relationship with the number of points N and the computational cost of each FFT. As a compromise between frequency resolution and computational cost, the SWE Bridge performs by default an FFT for each second of data, achieving a resolution of Δf = 1 Hz, which is sufficient for ocean sound analysis [36]. However, the frequency resolution may be adjusted by the user. The use of segments which are not power of 2 may slightly increment the computational cost, but a well-known frequency resolution is achieved in exchange. The time window set by the user is used to control the number of periodograms being averaged in order to reduce their variance. So, the user can select both frequency resolution and time window.

Both averaging methods are implemented within the SWE Bridge, the windowless Bartlett's method and the Welch method, using a Hann window with 50% overlap. Both approaches are depicted in Fig. 3.

1) Sound Exposure Level and Root Mean Square

Although SPL is the main parameter when measuring ocean sound, some other parameters may also be of interest, such as the Sound Exposure Level (SEL) and the Root Mean Squared (RMS) pressure. Since both parameters use similar calculations as the SPL, they are also implemented within the SWE Bridge to provide extra information at very little computational cost.

SEL is a measure of the integral of the square of the sound pressure over a stated time interval or event expressed in decibels [23]. This parameter is closely linked with the SPL value, but making the time interval explicit. It can be easily

derived from an SPL value using (14) [22].

$$SEL_T = SPL'_{f,N} + 10\log_{10}\left(\frac{T'}{T_0}\right)[dB\ 1\ \mu Pa^2 s] \quad (14)$$

Being T' the time window over which the SPL value was calculated (proportional to M) and T_0 = 1 second is the time reference.

The P_N is the RMS value of the pressure signal over a segment of length N, calculated using (15):

$$P_N = \sqrt{\frac{1}{N}\sum_N p(n)^2}[Pa] \quad (15)$$

In order to maintain the same time window as SPL and SEL values, several RMS values are averaged, as stated in (16).

$$\overline{P}_{N,M} = \frac{1}{M}\sum_M P_N[Pa] \quad (16)$$

Being $\overline{P}_{N,M}$ the mean of M RMS values. The SWE Bridge provides the averaged mean of several RMS values for two reasons: to maintain the same time window used in SPL and SEL calculations, and to prevent the use of very large buffer which may result in memory overflow and excessive computational cost.

2) Acquisition and Timestamping

In streaming mode, hydrophones usually send pressure samples grouped in packets. Within the SWE Bridge, the acquisition of incoming packets is performed through a configurable circular buffer. By default, the SWE Bridge defines a safe size for the buffer, but the user may adjust this parameter through the SensorML description to optimize the system's memory usage. A large circular buffer will waste a lot of memory, while a small buffer may result in overflows and loss of data.

Some hydrophones have an accurate internal real-time clock (RTC), providing precise timestamp information within the data stream's header, e.g. NeXOS A2 (see section V-C). The SWE Bridge can identify timestamps in the stream header and use them to maintain the time base. However, if a hydrophone does not provide timing information (e,g, NAXYS Hydrophone, see section V-A), incoming packets are timestamped upon arrival based on the platform's system clock. Thus, within the SWE Bridge the time base is maintained regardless of the hydrophone stream.

3) Missing Packets

Most digital hydrophones stream their samples grouped in packets using the User Datagram Protocol (UDP), which does not guarantee that all packets will be received. Data loss (packets not delivered) is intrinsic to UDP communications and cannot be avoided. Most digital hydrophones include a packet counter in their header, so the acquisition software can detect missing packets.

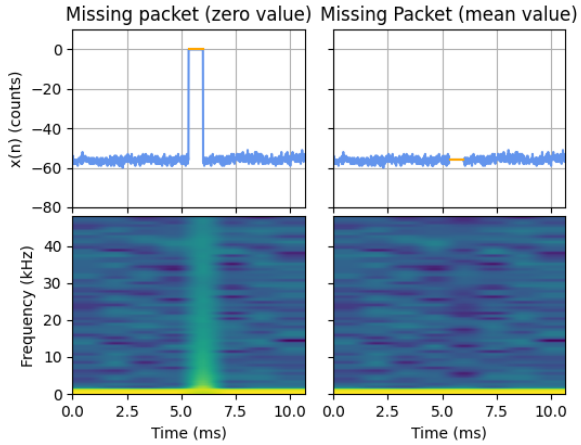


FIGURE 4. Spectral glitches due to missing packets. Left graphs show the pressure signal filled with zeros when a missing packet is detected (top) and its spectrogram (bottom), showing an artificial broadband glitch. Right graphs show the pressure signal filled with the mean value when a missing packet is received (top) and its spectrogram (bottom). It can be seen that the spectral glitch practically disappears when the mean value is used instead of zeros.

Since the time base has to be maintained, these missing packets have to be filled with null values, e.g. a packet with all values set to zero. However, these null values may induce frequency glitches and digital broadband distortion. In order to avoid these glitches, the SWE Bridge periodically calculates the mean value of the pressure signal during the last second. When a missing packet is detected, instead of filling the signal with zeros, the mean value is used. The spectral implications of these approach are shown in Fig. 4. This is critical to avoid erroneous data in hydrophones that have an offset in their signal or are measuring very low frequency signals.

IV. METADATA AND INTEROPERABILITY

Interoperability can be defined as the ability of two systems to exchange information and use the information that has been exchanged. It can be separated in two different layers: syntactic and semantic. The syntactic interoperability is the ability of two systems to understand their formats and interfaces, allowing the flow of information, while semantic interoperability focuses on providing unambiguous meaning to the information that has been exchanged.

Within the proposed architecture, metadata plays a vital role, giving contextual information about what is being measured, when and how. It also describes the communication's interface, protocol and signal conditioning characteristics. In other words, metadata is key to obtain both syntactic and semantic interoperability.

Relevant metadata to achieve both syntactic and semantic interoperability in hydrophone deployments have been identified and structured within a Hydrophone SensorML Template, depicted in Fig. 5. A hydrophone SensorML description file, encoded in XML format, can be easily processed by the different software components in the architecture, so

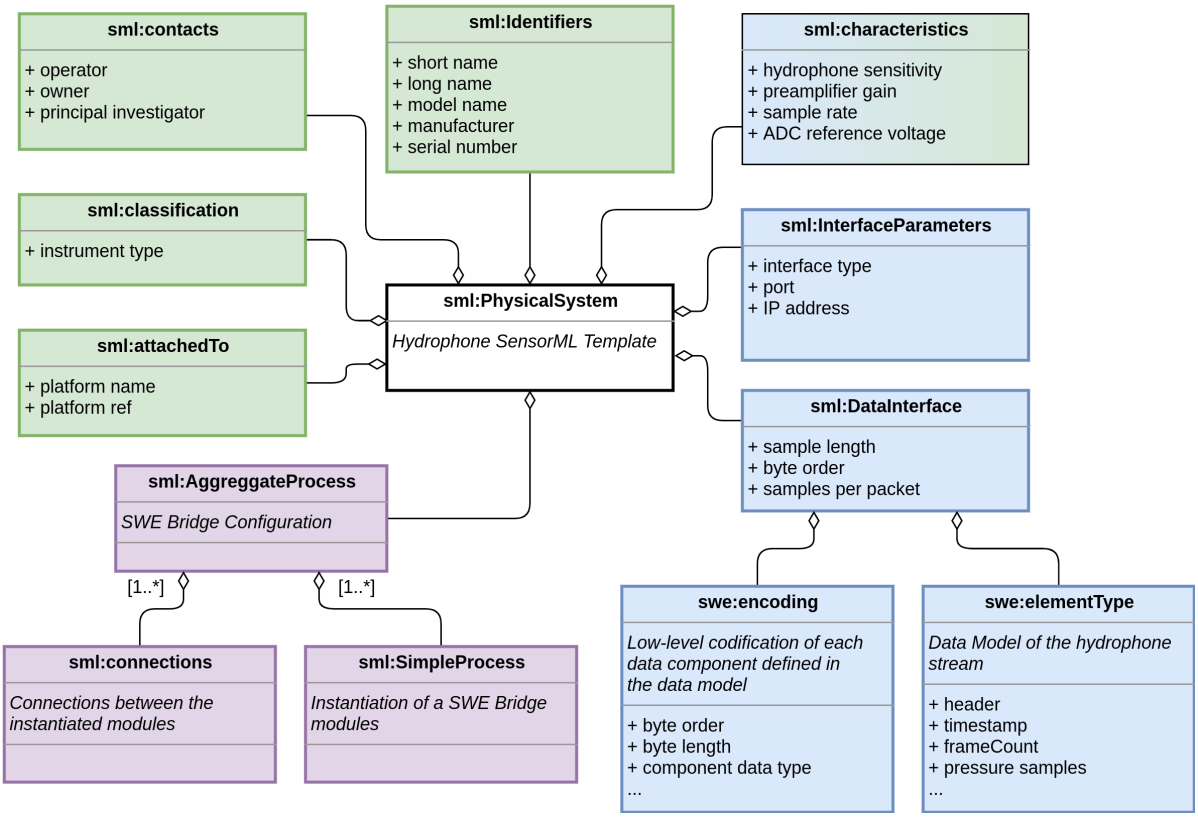


FIGURE 5. Diagram of the information contained within a Hydrophone SensorML Template. Green elements are used to provide contextual information for the measurements. Blue elements use mainly structures targeting syntactic interoperability. The *sml:characteristics* element is used in both, semantically and syntactically. Purple elements are used to configure the acquisition and data workflow within the SWE Bridge.

the metadata can be streamlined alongside data throughout the dataflow. In appendix B an example of a hydrophone SensorML description file can be found.

A. SYNTACTIC INTEROPERABILITY

The first step in any acquisition chain is to achieve syntactic interoperability between the acquisition software and the instrument, i.e. a hydrophone. Since manufacturers tend to use vendor-specific protocols and formats, achieving syntactic interoperability is not a trivial task. Usually an *ad hoc* driver is generated to interface a sensor to an acquisition system. However, within any driver there is a lot of implicit metadata to achieve syntactic interoperability, such as the communication protocol, the meaning of each field within a data stream, encoding, etc. All this information can be encoded in unambiguous way using the SensorML standard. A software component able to understand this standard could automatically configure its acquisition to achieve on-the-fly syntactic interoperability. Within the proposed architecture this component is the SWE Bridge.

However, hydrophone streams can be complex, containing headers, counters and acoustic data arranged in different ways. A detailed description of the stream has to be carefully organized and encoded. Signal conditioning characteristics such as transducer sensitivity, preamplifier gain and ADC parameters also need to be specified in order to transform dimensionless raw samples to pressure (see section III-A). Fig. 6 shows how the information within a hydrophone SensorML description is used by the SWE Bridge to configure the acquisition.

In order to communicate with a hydrophone, the first step is to define its communication interface: protocol (e.g. UDP or TCP), IP address and port number. Using this information, a communication link can be established and data streams start to flow. To process the incoming acoustic data, the low-level details of the stream must be known: packet length, byte order, sample width, etc.

At this point the SWE Bridge has interfaced the hydrophone on-the-fly and can acquire pressure samples for further processing. Its embedded underwater sound level algorithm can take incoming samples and provide relevant ocean sound measurements such as SPL, SEL and RMS values for each frequency band.

By default, the SWE Bridge applies this processing to be compliant with the MSFD descriptors following Task Group 11 recommendations. However, it is possible to adjust its parameters to fit different applications such as the frequency bands, time window and processing parameters. This configuration is also be included within the SensorML description file, so it contains both hydrophone metadata and information about how data is processed, from the transducer to the algorithm's output.

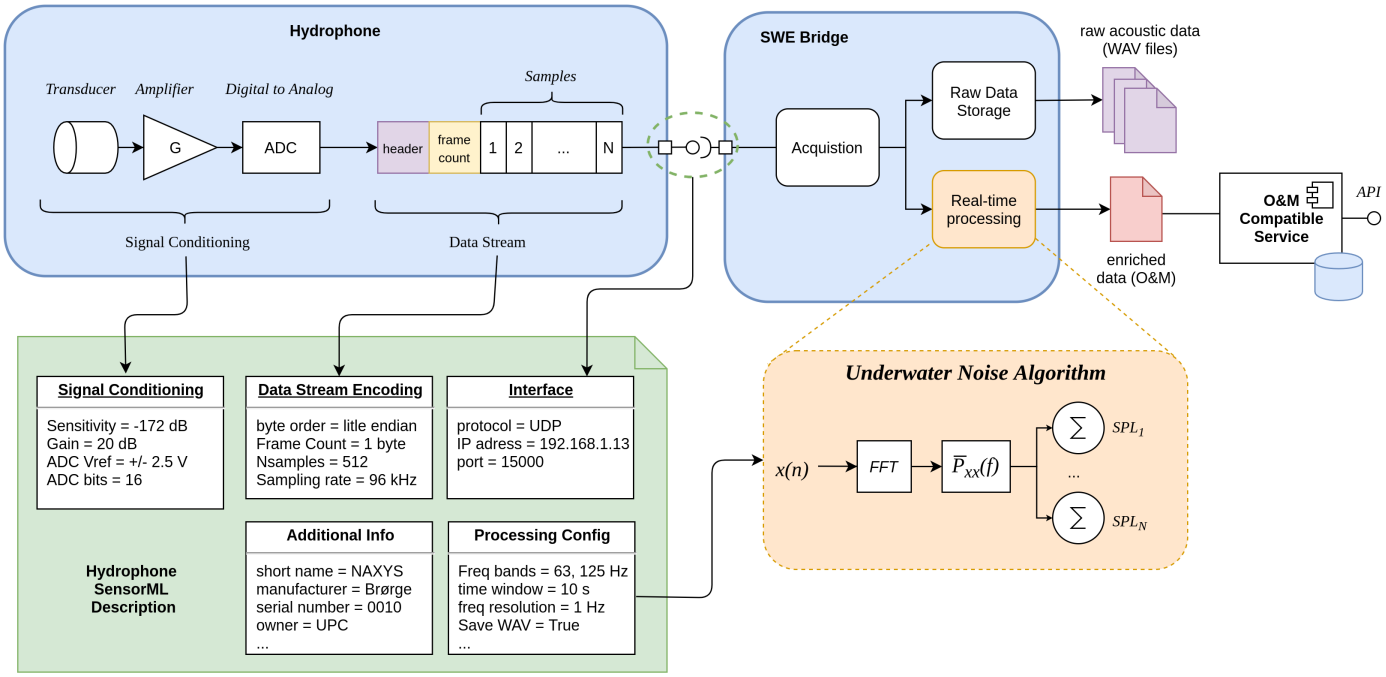


FIGURE 6. Metadata contained within a hydrophone SensorML deployment file and its role to achieve syntactic interoperability to establishing a data-flow. It includes signal conditioning metadata (e.g. sensitivity, gain, reference voltage), data stream encoding (e.g. byte order, sampling rate), communications interface (protocol, address, port, etc.), processing and storage options and contextual metadata (e.g. name, manufacturer, deployment coordinates).

B. SEMANTIC INTEROPERABILTIY

Semantic interoperability focuses on making the meaning of data and metadata explicit, with emphasis on machine to machine interactions. The overall goal is to provide machine understandable, standardized contextual information about the data, following the FAIR principles [26], [27].

The Hydrophone SensorML Template defines a minimum set of metadata elements to be included in a hydrophone SensorML description in order to provide accurate contextual information for the correct production of ocean sound measurements. SensorML is a very flexible and versatile standard, however this can also prove an issue, since the same information can be encoded in multiple ways [37]. In order to provide a semantically-robust metadata, controlled vocabularies are used within the proposed architecture.

The NERC Vocabulary Server version 2.0 (NVS2.0) provides access to standardized terms covering a broad spectrum of disciplines relevant to the oceanographic and earth observing community [38]. These terms are organized collections of concepts, named vocabularies. Each concept has its own universal resource identifier (URI) that resolves, after content negotiation, in a self-descriptive resource description format (RDF) or an HTML page depending if the request was made by a human or a machine. NVS2.0 also holds a set of vocabularies specifically targeting SensorML terms, developed by the SWE Marine Profiles communities [39]. Table 1 shows the minimum set of terms from the NVS2.0 used in the hydrophone SensorML template.

NVS2.0 does not provide specific terms for some passive acoustics properties such as hydrophone sensitivity, pre-

TABLE 1. Minimum set of terms from the NERC Vocabulary Server 2.0 used within the Hydrophone SensorML Template.

Vocabulary	Term	Label
W06	CLSS0002	instrument type
W07	IDEN0006	short name
W07	IDEN0002	long name
W07	IDEN0003	model name
W07	IDEN0012	manufacturer
W08	CONT0003	operator
W08	CONT0004	principal investigator
W08	CONT0002	owner
P07	CFSN0310	Sound Pressure Level in water

TABLE 2. Minimum set of terms used from the Integrated Ocean Observing System Passive Acoustics Conventions vocabulary.

Vocabulary	Term
IOOS PAM Conventions	hydrophone_sensitivity
IOOS PAM Conventions	sample_rate
IOOS PAM Conventions	preamplifier_gain

amplifier gain and sample rate. As an alternative for these terms the Integrated Ocean Observing System (IOOS) Passive Acoustic Monitoring (PAM) Conventions vocabulary has been adopted [40]. Table 2 shows the minimum set of terms from the IOOS PAM conventions vocabulary used within a hydrophone SensorML template. However, not all terms can be found in neither vocabularies, e.g. ADC's reference voltage.

The processed data generated by the SWE Bridge, alongside with the semantically-enhanced metadata from the SensorML are combined into O&M files, which can be injected in standard services such as SOS [28].

C. EMBEDDING METADATA IN WAV FILES

Alongside the semantically-enriched O&M data files, the SWE Bridge may also generate raw acoustic recordings using the Waveform Audio File Format (WAV). This format is an extension of the Resource Interchange File Format (RIFF) focusing on commercial audio.

Due to its versatility and flexibility, the WAV format has been broadly used in passive acoustics for hydrophone recordings. However, it does not consider any standardized mechanism to embed metadata. A WAV file without its associated metadata may be useless since the calibration and contextual information are not known (hydrophone sensitivity, location, timestamp, etc.).

In order to overcome the lack of metadata, the SWE Bridge uses the ID3 tagging system. This informal standard takes advantage of the RIFF's chunk-based design, adding a chunk of metadata alongside the audio data while maintaining the format compatibility [41]. ID3 provides a list of items that can be included, such as author, song title, genre, composer, etc.

Although it focuses on commercial audio and it is not directly applicable to underwater acoustics, the ID3 tagging system includes the option to add user-defined tags. Within the SWE Bridge software, these user-defined tags are leveraged to include key-value pairs with all the relevant information contained within the SensorML hydrophone description, such as hydrophone name, model, sensitivity, deployment position, etc. Digital audio workstations and audio players can easily access to the embedded metadata. Using this workaround all the metadata described in section IV is embedded in the WAV file, maintaining its compatibility with WAV and RIFF formats.

V. USE CASES

A. NAXYS HYDROPHONE AT OBSEA

OBSEA Expandable Seafloor Observatory is a cabled, multiparametric observing platform located 4 km off the coast of Vilanova i la Geltrú (Barcelona, Spain) at a depth of 20 meters. Since its deployment in 2009 it has been continuously acquiring data from numerous variables such as temperature, salinity, pressure, air temperature and underwater sound [42]. OBSEA is equipped with a NAXYS Ethernet 02345 hydrophone, streaming acoustic data. Since 2020 it is offering ocean sound measurements in real-time using the proposed ocean sound architecture, as shown in Fig. 7. It provides valuable data for the assessment of long-term underwater noise trends in the eastern Mediterranean Sea, compliant with the requirements of the MSFD.

The SWE Bridge universal driver is deployed within the platform, using the information contained within the hydrophone's SensorML description file to setup the data acquisition. It automatically processes the incoming data stream, transforming from raw samples to pressure values. Its embedded underwater sound algorithm computes SPL, SEL and RMS values with a time window of 10 seconds. It uses the Welch method with a Hann window and an overlap of 50 % (see section III-C). These parameters are calculated for the 63, 125 and 2000 Hz third-octave bands and the whole bandwidth (up to 48 kHz). The hydrophone's specifications and the processing setup is shown in table 3.

TABLE 3. NAXYS Ethernet Hydrophone 02345 specifications

Hydrophone Parameter	**Value**
sample rate	96000 Hz
hydrophone sensitivity	-192 dB re V/µPa
preamplifier gain	20 dB
ADC reference voltage	± 2.5 V
ADC number of bits	16
Processing Parameter	**Value**
frequency resolution	1 Hz
time window	10 s
overlap	50 %
window function	Hann
third-octave bands	63, 125, 2000 Hz

The algorithm result is stored in O&M files and sent to OBSEA's Sensor Observation Service. Data is available in real-time through the Helgoland Client at http://sos.obsea.es/client or directly through the SOS interface http://sos.obsea.es/sos/. The hydrophone's SensorML description is also available via SOS interface.

Raw acoustic data in WAV format is also generated by the SWE Bridge to allow further analysis and data validation. Since OBSEA is a cabled observatory and does not present communications constraints, the generated WAV files are sent in real-time to the OBSEA's ERDDAP server [30], available at http://erddap.obsea.emso.eu.

ERDDAP is a data server that provides a simple, consistent way to serve scientific data on the web. Alongside with its web interface it also provides a RESTful API allowing humans and/or machines to interact with data programmatically. Unlike SOS, it does not use strong semantics for metadata, but its ability to manage different files and formats makes it a perfect candidate to serve large datasets with heterogenous data. OBSEA's ERDDAP server contains WAV datasets and SPL timeseries. Using ERDDAP and SOS the data is shared with data portals and services, such as EMODnet Physics [43].

Fig. 8 shows ocean sound data at OBSEA from 1st of May until 15th of September 2020, acquired using the proposed metadata-driven architecture. SPL data is averaged in periods of 6 hours for visualization purposes.

Ocean sound has a high variability, as it can be seen in the dataset. The arithmetic mean of large SPL dataset is heavily influenced by short, high-intensity events. In other words, it is highly sensible to outliers in the underwater sound level distribution [44] [6]. When assessing underwater sound levels, different averaging metrics can produce widely differing levels, which may result in misinterpretation of ocean sound data [45]. To complement the information provided by the arithmetic mean, additional metrics are provided at table 4. The mode and the median provide a better estimate of the expected SPL level during most of the time, and the L90 percentile shows the level which is not exceeded in 90 %

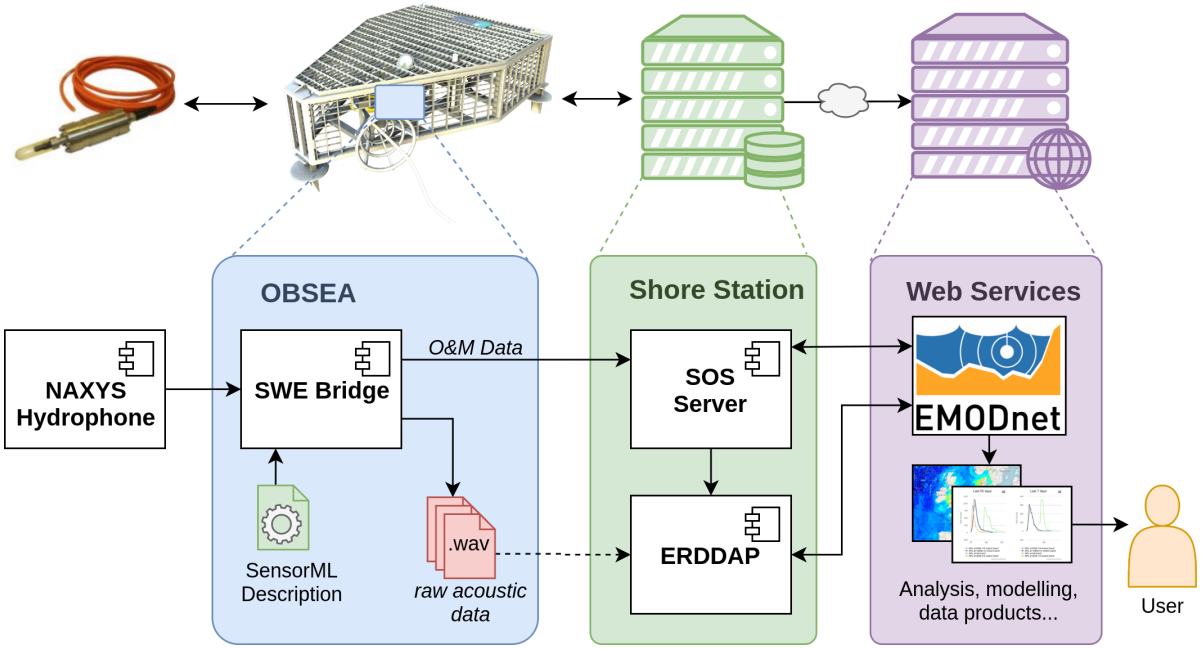


FIGURE 7. Ocean sound measurement dataflow at OBSEA. The SWE Bridge interfaces the NAXYS hydrophone based on the information on its SensorML description and generates two outputs: underwater sound levels encoded in O&M and acoustic recordings, sent to a SOS and ERDDAP servers respectively. This data is then shared data portals and repositories such as EMODnet Physics.

TABLE 4. Values of Sound Pressure Level full bandwidth (all) and 1/3 octave bands centered at 63, 125 and 2000 Hz at OBSEA. All values are in dB re 1 µPa with a time window of 10 seconds.

metric	all	63 Hz	125 Hz	2000 Hz
mode	96.04	77.13	75.83	74.17
median	97.46	77.96	77.55	75.74
L90	103.81	86.79	88.42	81.89
mean	105.12	88.04	93.04	84.21

of the time. As it can be seen, the mean of the SPL values is higher than the L90 percentile (up to 95 % in the 125 Hz third-octave band timeseries).

The spike observed on the 20th of July corresponds with a major maintenance operation at OBSEA using a large support vessel. The increment of the underwater sound level is caused by the proximity of the vessel's engine.

Weekly periodic oscillations within the time series can be seen, especially in May and June. These oscillations are probably due to the increment of human activities, during workdays (e.g. fishing) and a decrease during weekends. However, in July and August these fluctuations are less evident due to the increase of recreational boating

From the probability density it can be seen that the sound levels at 2000 Hz are very stable, while low-frequency sound (63 and 125 bands) have a larger variability. This may be induced by the proximity of the port of Vilanova i la Geltrú, as low frequency noise produced by fishing vessels passing by are acquired. Higher frequencies are attenuated much more rapidly than lower frequencies, thus the 2000 Hz band is much less affected by distant events (e.g. shipping), presenting a narrower distribution.

B. EGIM PASSIVE ACOUSTICS RECORDINGS

Although real-time ocean sound processing is the main goal of the presented architecture, it is also capable of analyzing previously acquired data. The SWE Bridge can access acoustic recordings in WAV format and generate processed ocean sound datasets in O&M or CSV formats.

In order to illustrate this capability, recordings from a hydrophone deployed within an EGIM (EMSO Generic Instrument Module) have been processed [46]. The deployment was performed at the Plataforma Oceánica de Canarias (PLOCAN) in the Canary Islands, Spain, during June and July 2019. The EGIM sensor system contained an Ocean Sonics icListen HF hydrophone continuously recording data in its internal memory. The overall dataset was retrieved after instrument recovery and processed using the SWE Bridge. The hydrophone specifications (table 5) were encoded within a SensorML file and used by the SWE Bridge to convert the raw acoustic dataset to SPL, as shown in Fig. 9. The resulting dataset of SPL values was uploaded to PLOCAN's ERDDAP service, where it is publicly available (http://erddap.plocan.eu/erddap).

Fig. 10 shows the result of a very unstable underwater soundscape. Very intense low frequency noise (below 20 Hz) has been observed in the dataset. In order to reduce the spectral leakage from low frequencies the Welch method using a Hann window has been selected, with time window of 10 seconds and a Δf of 1 Hz and an overlap of 50% (see section III-C). The metrics of the overall dataset are shown in table 6.

PLOCAN's test site is approximately 1 km away from Taliarte's port, a relatively small fishing port. Small ships

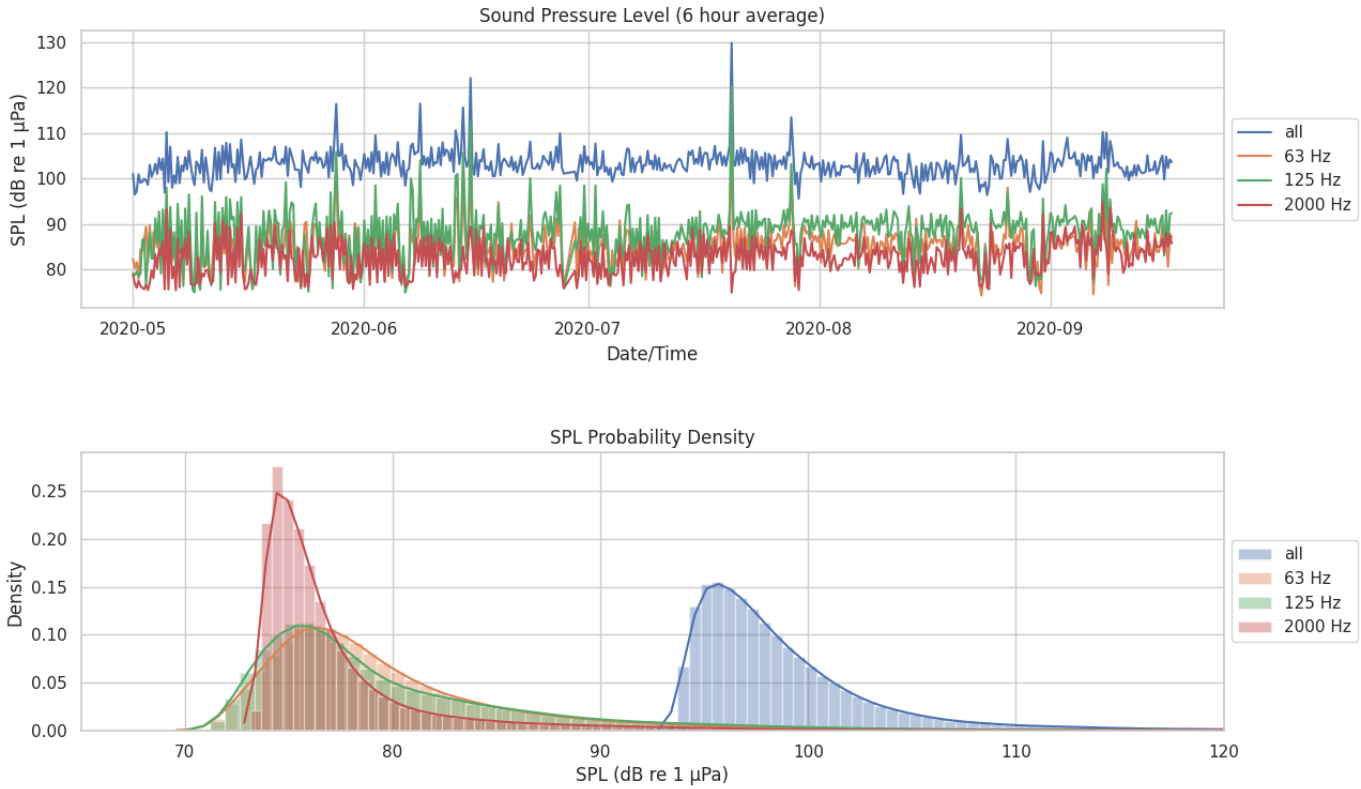


FIGURE 8. Sound Pressure Level (SPL) at full bandwidth, 63, 125 and 2000 Hz third-octave bands at OBSEA from May 1st to September 15th 2020, averaged in periods of 6 hours (top), and SPL histogram and estimated probability distribution function (bottom).

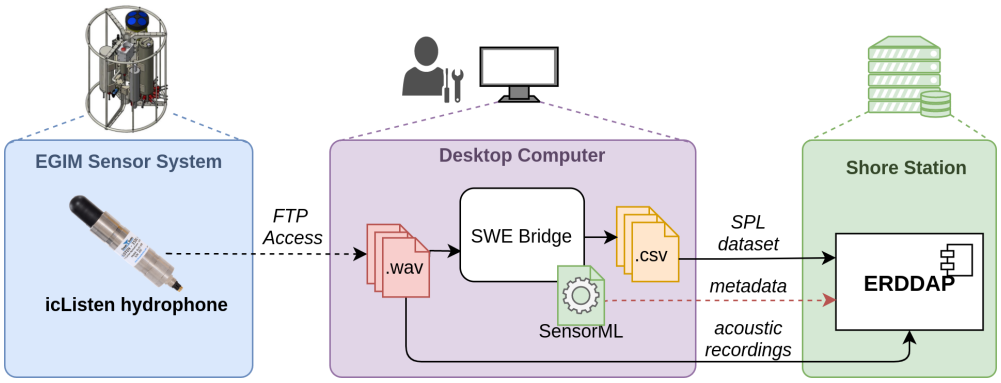


FIGURE 9. Ocean sound measurement dataflow of the EGIM sensor system at PLOCAN. Internally recorded data was stored on-board and downloaded via FTP. Afterwards the SWE Bridge processed the acoustic recordings, generating a Sound Pressure Level dataset in CSV format. The resulting dataset and its metadata is injected to an ERDDAP service.

very close to the hydrophone may be the cause of spikes observed in the dataset. The low-frequency noise may be induced by shipping from the port of Las Palmas de Gran Canaria (a major commercial port), approximately 20 km away from the deployment site and possibly the logistics required around a wind energy converter deployed close to PLOCAN.

Although there is a clear influence of shipping, the changes in the trend can be clearly observed in the dataset in all frequency bands. Wind speed data collected from a weather station at PLOCAN also shows a correlation with the background trend (see Fig. 11), thus possibly stemming from the added sound energy generated by wind-driven wave-breaking processes.

C. A2 HYDROPHONE ARRAY

The A2 Hydrophone Array is a digital passive acoustic transducer array designed for sound source localization (SSL) and tracking. It is composed of 4 slave hydrophones, called A2 hydrophones, streaming data to a master unit which processes

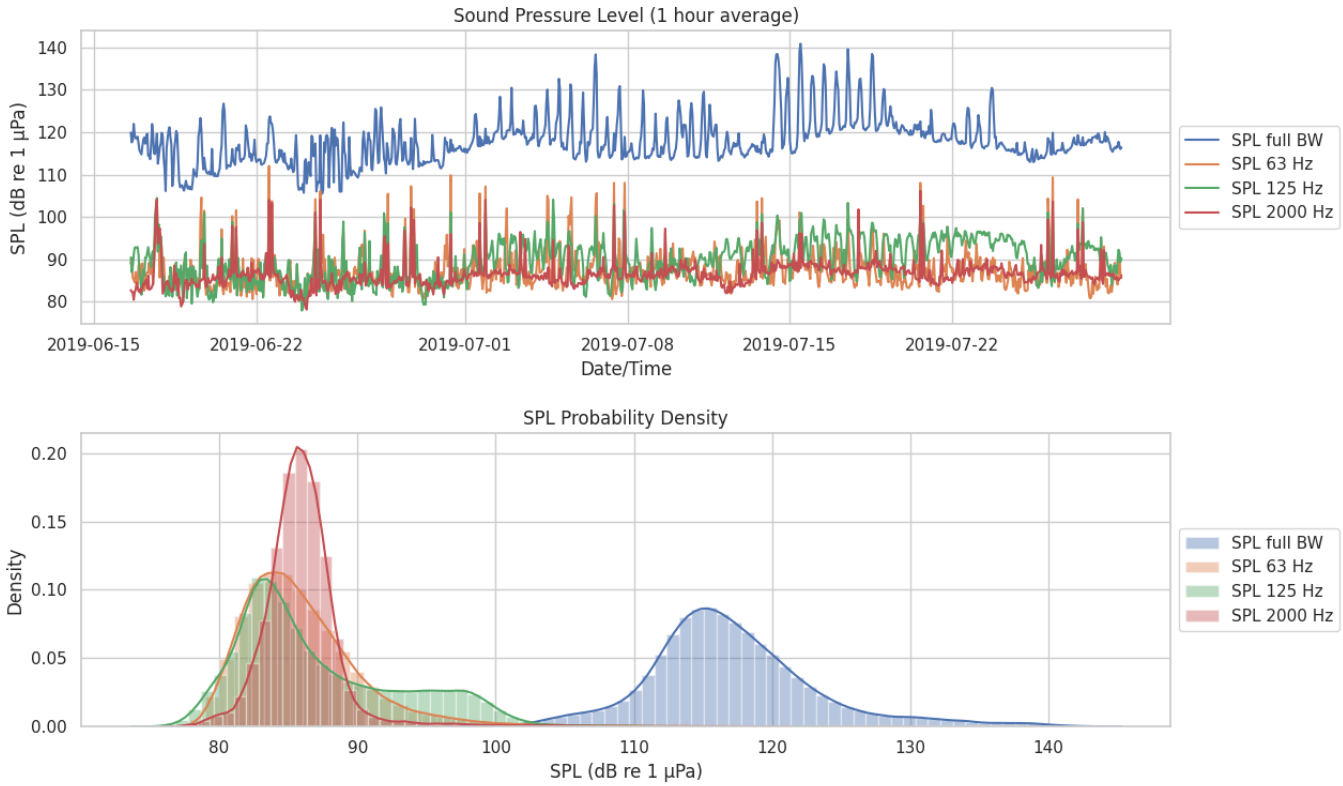


FIGURE 10. Sound Pressure Measurements at full bandwidth, 63, 125 and 2000 Hz third-octave bands at PLOCAN using an icListen hydrophone during June and July 2019, averaged in periods of 1 hour for visualization purposes (top), and SPL histogram and estimated probability distribution function (bottom).

TABLE 5. icListen HF hydrophone specifications

Hydrophone Parameter	**Value**
sample rate	8000 Hz
hydrophone sensitivity	-169 dB re V/µPa
ADC reference voltage	± 3 V
ADC number of bits	24
Processing Parameter	**Value**
frequency resolution	1 Hz
time window	10 s
window function	Hann
overlap	50 %
third-octave bands	63, 125, 2000 Hz

TABLE 6. Metrics of the Sound Pressure Level full bandwidth (all) and third-octave-bands centered at 10, 63, 125 and 2000 Hz at PLOCAN during June and July 2019. All values are in dB re 1 µPa.

metric	**all**	**10 Hz**	**63 Hz**	**125 Hz**	**2000 Hz**
mode	115.99	89.96	84.36	84.38	86.19
median	116.33	90.40	85.15	85.28	85.82
L90	124.28	99.00	91.24	96.74	88.42
mean	123.37	107.53	92.80	92.53	89.17

the acoustic data in real-time. Therefore, the main capability of A2 is to provide directional sound source information for hydro-acoustic surveys [15]. Fig. 12 shows the block diagram of the A2 hydrophone array.

Time synchronization between the master unit and the slave units (A2 hydrophones) is accomplished by implementing the IEEE 1588 Precision Time Protocol (PTP) standard. This standard defines a network protocol enabling accurate and precise synchronization of the real-time clocks of devices in networked distributed systems, achieving a precision below the micro-second. The acoustic data is transmitted from the slave to the master by means of the Real-time Transport Protocol (RTP).

The A2 Hydrophone Array can be equipped with positioning sensors (pan, tilt, and compass) to allow the measurement of its geo-referenced position. The device can also receive relevant oceanographic parameters (sound velocity, temperature, depth, time) via Ethernet, in order to optimize the algorithms.

Within this work, the A2 hydrophone array's capabilities have been extended to provide ocean sound measurements alongside sound source localization. The proposed architecture for ocean sound measurement has been seamlessly integrated within the A2 hydrophone array, as depicted in Fig. 13. Taking advantage of the SWE Bridge's functionality, raw acoustic recordings for each hydrophone are also stored on-board for further analysis and data validation.

The A2 master unit, based on an embedded Linux single-board computer, receives real-time data from the slaves hydrophones. Thus, an instance of the SWE Bridge for each slave hydrophone has been deployed within the master unit.

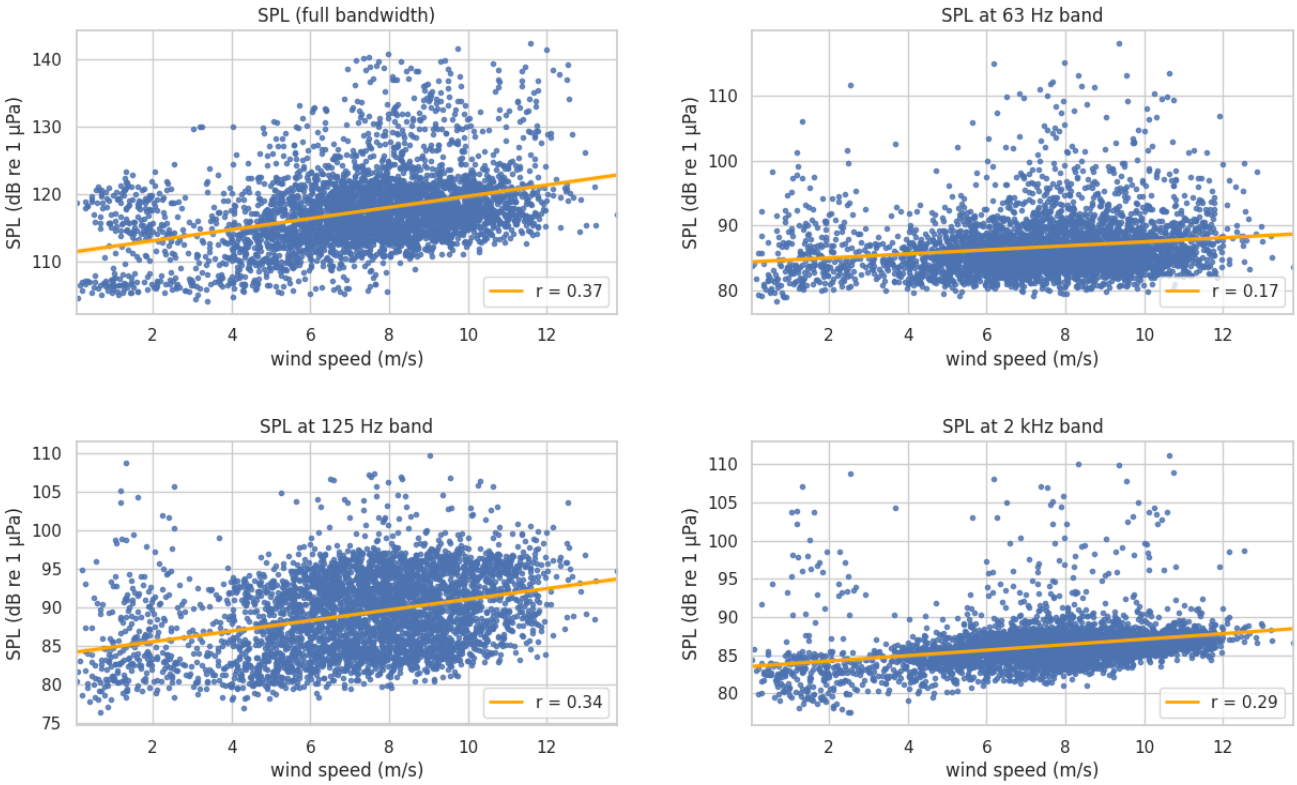


FIGURE 11. Linear regression between Sound Pressure Level (SPL) and wind speed and its correlation coefficient *r* at PLOCAN during June and July 2019. The linear regression analysis shows a clear correlation between both variables, although the outliers in SPL measurements induced by others sources (e.g. shipping) reduce the correlation coefficient *r*.

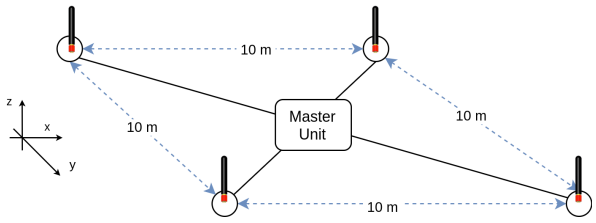


FIGURE 12. A2 hydrophone array diagram.

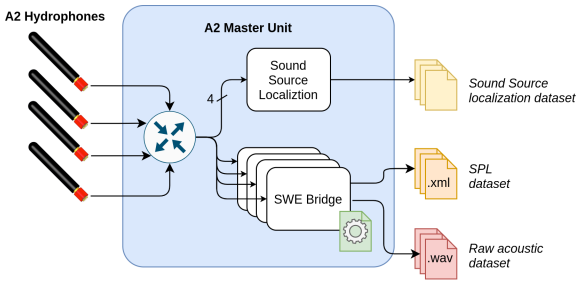


FIGURE 13. A2 hydrophone array extended with the SWE Bridge for ocean sound monitoring.

The acoustic streams coming from the slave hydrophones are redirected to the SSL algorithm and to SWE Bridge instances. Each instance, processing data of a slave hydrophone, has its own SensorML hydrophone description file, used to setup the acquisition. Those files are identical with the exception of the communication's interface configuration, serial number and their calibration information. As output, four underwater sound levels and acoustic recordings datasets are generated, one per slave hydrophone.

Successful lab tests were performed and a deployment is scheduled by the beginning of 2021.

VI. CONCLUSIONS AND FUTURE WORK

Within this work a universal architecture for ocean sound monitoring is proposed. A generic, real-time algorithm implementation for *in situ* underwater sound monitoring has been presented, following best practices on underwater noise measurement methodologies. The intrinsic constraints of state-of-the-art observation platforms have been carefully considered in order to achieve an efficient, generic and cross-platform implementation.

Open-source and open standards are used to ensure reusability and universality of such architecture, regardless of the underlying components (hydrophone model, observations platform, etc.). Interoperability challenges at both syntactic and semantic level have been thoroughly analyzed and a solution based on the SensorML standard has been proposed. At the syntactic (operational) level, a characterization of the

hydrophone leads to a seamless integration within the proposed architecture by abstracting the sensor characteristics. At the semantic level, relevant contextual information has been discussed and encoded using controlled vocabularies to achieve rich and coherent metadata, providing the building blocks for a FAIR data management.

The discussed metadata solution does not only provide contextual information, but effectively manages the acquisition system operation. Since all the acquisition and processing steps are explicitly encoded within the presented metadata solution, the whole acquisition and processing chain can be easily replicated.

In order to demonstrate its flexibility, the architecture has been applied to three different scenarios: real-time monitoring in a cabled observatory, the analysis of acoustic recordings and as the enhancement of a hydrophone array. In all cases the architecture effectively managed to abstract underlying hardware / software characteristics and underwater sound levels measurements were successfully acquired and processed.

One of the weakness of the proposed architecture is the need to generate large SensorML metadata files, which are soft-typed and based on a template. A future line of work would be to define a normative hydrophone description profile, so the generated metadata files could be validated against a schema. Moreover, this schema could be integrated into existing SensorML editing tools to ease its generation and maintenance.

Future work in metadata could move beyond sensor deployment and focus on sensor calibration and operational history, providing traceability throughout the whole instrument's life-cycle.

As a future line of research, the proposed architecture could expanded to other underwater acoustics applications. Since hydrophone integration and data pre-processing steps have been addressed within this work, new algorithms can be easily integrated, such as sound source recognition, marine species detection, bio-acoustics, etc.

.

APPENDIX A ALGORITHM COMPARISON

Although the typical approach to calculate SPL at different band frequencies is the spectral analysis by means of FFT, there are other approaches that may achieve the same result. In this section, three methods were tested to obtain a computationally efficient SPL algorithm for an arbitrary number of 1/3 octave band frequencies. The tested algorithms were a filter bank, the FFT-based algorithm and the Goertzel's algorithm.

The filter bank approach to calculate the third-octave SPL levels involves decimating and filtering, as shown in Fig. 14. Generally, hydrophones have a broad bandwidth, up to tens or hundreds of kHz. Thus, its sampling rate is usually much higher than the frequencies of interest. In order to save computational resources and to ensure the filter stability, the incoming signal needs to be decimated.

The decimation process involves low-pass filtering to avoid aliasing, and down-sampling. Since underwater sound level algorithms are usually calculated for low-frequency third-octave bands, the incoming signal may be over-sampled by several orders of magnitude over the Nyquist rate. In this case, decimating the signal in a single stage may result in a very costly and complex filtering arrangement, so it may be more efficient to decimate the incoming signal in multiple stages [47]. The optimal number of decimation stages needs to be calculated at run-time based on the hydrophone's sampling rate and the highest cut-off frequency of the third-octaves bands.

After being decimated, the pressure signal is filtered by a band-pass filter. Ideally such a filter would completely eliminate the energy in the rejection-band without affecting the power in the band-pass. In practice, it should have small ripple at the band-pass and a fast roll-off in the rejection bands to minimize its influence on the signal. When designing a filter there is a trade-off between the computational cost and the filter performance. Generally various frequency bands are desired; thus, a filter bank is required.

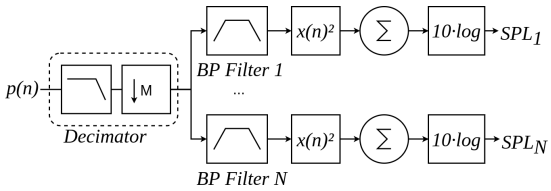


FIGURE 14. Block diagram of a SPL algorithm for multiple band frequencies using a filter bank.

Once the signal has been filtered, the SPL value for a specific frequency band can be calculated using (4) over the filtered pressure signal.

On the contrary to the filter bank approach, the power spectral density of the pressure signal can be estimated by means of a Discrete Fourier Transform (DFT), as discussed in section III. To calculate the DFT, two different implementations where evaluated, the well-known Fast Fourier Transform (FFT) and the Goertzel's algorithm.

The FFT is an optimized implementation of the DFT for data segments with power of 2 length. On the contrary, Goertzel's algorithm is much slower DFT implementation, but it has the ability to independently calculate DFT terms. Goretzel's algorithm can be useful when only a small number of DFT bins M are required, i.e. $M < log_2 N$ [31]. In this particular application, the overall spectrum is not desired, just the bins containing frequencies within the bands of interest. So, it has been considered for ocean sound measurements.

All three SPL algorithms where implemented and tested to assess their performance: a filter bank (multi-stage decimation and 5th order elliptic band-pass filters), another FFT-based and the third one based on the Goertzel's algorithm. Several tests were executed to assess their performance in terms of required memory and execution time under different conditions, such as the number of samples in the signal

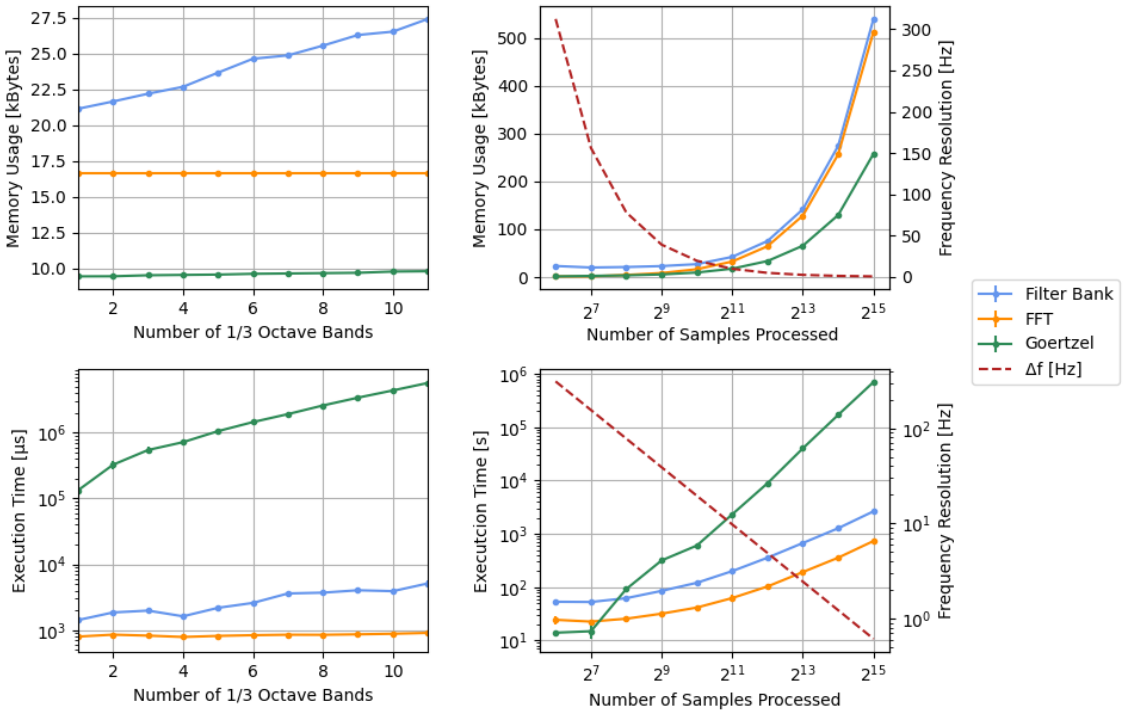


FIGURE 15. Performance test of three different SPL algorithms based on their 1/3 octave estimation method: filter bank, FFT and Goertzel's algorithm. The left graphs show the memory usage (top) and the execution time (bottom) depending on the number of third-octave bands calculated, starting from the band centered at 63 Hz (N is set to 4096). The right graphs show the memory usage (top) and execution time (bottom), depending on the number of samples in the time window (only 63 and 125 Hz third-octave bands). The dashed red line in rights graphs represent the frequency resolution Δf of the DFT for each value of N. The sampling frequency is set to 20 kHz.

and number of third-octave bands calculated. Although the MSFD only specifies two 1/3 octave bands to be monitored (63 and 125 Hz), it has been suggested to extend the monitoring range up to 20 kHz [3]. Thus, the computational impact of increasing the number of band frequencies calculated has been assessed.

The algorithms where implemented using the python3 programming language. Although it is a high-level scripted programming language and most of the complex operations are done under-the-hood, it is a good starting point to assess the feasibility of each algorithm and have a rough estimate of their computational cost. The results of these tests are shown in Fig. 15.

The filter bank memory usage grows rapidly when more third-octave bands are calculated (Fig. 15, top left). The filter bank is not heavily affected by the increase of the signal's length N (Fig. 15, bottom right), however it is significantly affected when the number of third-octaves bands are increased, since a different filter has to be applied for each band (top right).

The Goertzel's algorithm has a very limited memory usage and a very good performance when applied to small number of samples N. However, its execution time exponentially grows as N or the number of third-octave bands is increased. The dashed red line shows the Δf achieved with the DFT (both Goertzel and FFT), which is inversely proportional to N. The Goertzel's algorithm is only faster than the other algorithms when Δf is in the order of tens of Hz, which is not acceptable for the intended applications. When accurate frequency resolution is required, the algorithm shows very poor performance (note the axis logarithmic scale). Thus, this algorithm is not suited for this particular application.

Although FFT has an average memory usage with respect to N, it shows very good performance in terms of execution time. Moreover, its memory usage is constant regardless of the number of third-octave bands calculated. Thus, the FFT-based SPL algorithm has been selected and implemented within this work.

APPENDIX B HYDROPHONE SENSORML DESCRIPTION

This section provides an example of a hydrophone SensorML description, using OBSEA's NAXYS hydrophone as an example (see section V-A). The following XML snippets provide examples on how to define specific parts of the document. URLs are shortened to improve readability. The entire document is available online at http://sos.obsea.es/sensorml/naxys.xml.

Table 7 shows the metadata included within the NAXYS SensorML description. Some of the components such as hydrophone data stream and SWE Bridge configuration are not included in the table, but are discussed later in this section.

TABLE 7. Metadata included in the NAXYS Hydrophone SensorML description.

Term	Value
sml:classification	
instrument type	SDN:L05::369 [hydrophone]
sml:identifiers	
short name	NAXYS Hydrophone
long name	NAXYS Hydrophone at OBSEA
model name	NAXYS Ethernet Hydrophone 02345
manufacturer	Bjørge, AS
serial number	0010
sml:contacts	
operator	Enoc Martínez
owner	Universitat Politècnica de Catalunya
principal investigator	Joaquín del Río
sml:characteristics	
hydrophone sensitivity	-192 (dB re 1 V/µPa)
preamplifier gain	20 (dB)
sample rate	96000 (Hz)
ADC reference voltage	2.5 (V)
sml:InterfaceParameters	
interface type	UDP
port number	15000
IP	192.168.1.113
sml:attachedTo	
title	OBSEA
reference	http://sos.obsea.es/sensorml/obsea.xml
sml:position	
latitude	41.1819 (degree north)
longitude	1.7527 (degree east)
depth	20 (m)

A. IDENTIFICATION, CLASSIFICATION AND CAPABILITIES

Identification and classification sections provide a URI to a controlled vocabulary (definition attribute in the example below), a label with a human-readable description of the term and a value. This value can be either a text string (e.g. NAXYS Hydrophone) or another URI to a controlled vocabulary where the info is described (e.g. description of the sensor type in NVS2.0's L05 vocabulary). The following snippet shows how to define the short name parameter:

```
<sml:identifier>
  <sml:Term definition="http://vocab.../IDEN0006/">
    <sml:label>short name</sml:label>
    <sml:value>NAXYS Hydrophone</sml:value>
  </sml:Term>
</sml:identifier>
```

In the capabilities section the electrical properties of the hydrophone are described. Using the SWE Common Data Model standard it is possible to provide unambiguous definition of the data type and the associated units. The following snippet shows how to describe the hydrophone sensitivity:

```
<sml:capability name="hydrophone_sensitivity">
  <swe:Quantity
     definition="http://mmisw.../hydrophone_sensitivity">
     <swe:label>hydrophone sensitivity</swe:label>
     <swe:uom code="dB_re_uPa"
       xlink:href="http:/vocab..../UDBL"/>
     <swe:value>-192</swe:value>
  </swe:Quantity>
</sml:capability>
```

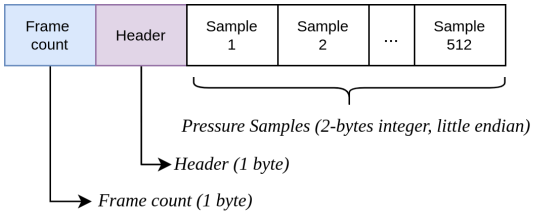


FIGURE 16. NAXYS Hydrophone stream, composed by a frame counter byte (0-255), a header byte and 512 pressure samples encoded as 2-byte integers in little endian.

B. CONTACTS

The contacts section provides information about the people and organization involved. SensorML does not describe a specific set of terms for these, but it uses the ISO 19115 to provide information about the contacts and their role.

C. INTERFACE PARAMETERS

In order to define unambiguously the interface details such as IP address and port number the *sml:interfaceParamaters* within a *sml:parameter* is used.

```
<sml:interfaceParameters>
  <swe:DataRecord>
    <swe:field name="portType">
      <swe:Category>
        <swe:label>port type</swe:label>
        <swe:value>UDP</swe:value>
      </swe:Category>
    </swe:field>
    <swe:field name="portNumber">
      <swe:Count>
        <swe:label>port number</swe:label>
        <swe:value>15000</swe:value>
      </swe:Count>
    </swe:field>
    <swe:field name="IP">
      <swe:Count>
        <swe:label>IP address</swe:label>
        <swe:value>192.168.1.113</swe:value>
      </swe:Count>
    </swe:field>
  </swe:DataRecord>
</sml:interfaceParameters>
```

D. ATTACHED TO

The *sml:attachedTo* element is used to provide information about the observing platform where the sensor is installed. According to the SensorML standard, the *xlink:title* attribute is used to specify the identifier of the hosting system while the *xlink:href* should point to a SensorML description of such platform:

```
<sml:attachedTo xlink:title="OBSEA"
  xlink:href="http://sos.obsea.es/sensorml/obsea.xml"/>
```

E. DATA STREAM

One of the most complicated aspects of a hydrophone SensorML description is complex streams in binary format. Although it is not trivial, the SWE Common Data Model Standard provides a good framework to encode such streams

[48]. The NAXYS hydrophone sends streams periodically, each one with 1026 bytes. As depicted in Fig. 16, each stream has a frame counter to detect missing packets, a header byte containing configuration information (sampling rate and preamplifier gain) and 512 pressure samples, arranged in signed 2-byte integers. The following SensorML excerpt shows how the NAXYS hydrophone stream has been modelled using the *swe:DataStream* element:

```
<swe:DataStream>
  <swe:elementType name="dataModel">
    <swe:DataRecord>
      <swe:field name="frameCount">
        <swe:Count/>
      </swe:field>
      <swe:field name="header">
        <swe:Count/>
      </swe:field>
      <swe:field name="array">
        <swe:DataArray>
          <swe:elementCount xlink:href="#arrayCount"/>
          <swe:elementType name="samples">
            <swe:Count
              definition="http://vocab.../CFSN0310/"/>
          </swe:elementType>
        </swe:DataArray>
      </swe:field>
    </swe:DataRecord>
  </swe:elementType>
  <swe:encoding>
    <swe:BinaryEncoding byteOrder="littleEndian"
      byteEncoding="raw" byteLength="1026">
      <swe:member>
        <swe:Component
          dataType="http://www.opengis.net/.../unsignedByte"
          ref="dataModel/header"/>
      </swe:member>
      <swe:member>
        <swe:Component
          dataType="http://www.opengis.net/.../unsignedByte"
          ref="dataModel/frameCount"/>
      </swe:member>
      <swe:member>
        <swe:Component
          dataType="http://www.opengis.net/.../signedShort"
          ref="dataModel/array/samples"/>
      </swe:member>
    </swe:BinaryEncoding>
  </swe:encoding>
  <swe:values/>
</swe:DataStream>
```

The *swe:DataStream* element is used to encode the stream of the hydrophone. It contains two main components: the data model (*swe:elementType*) and the low-level encoding details (*swe:encoding*). Within the data model three fields are declared, the frame count, the header byte and an array of pressure samples. The number of pressure samples within the array is declared as a parameter outside the data stream and referenced in the data model:

```
<sml:parameter name="arrayCountParameter">
  <swe:Count id="arrayCount">
    <swe:value>512</swe:value>
  </swe:Count>
</sml:parameter>
```

The *swe:encoding* defines the low-level encoding details for the stream. These details include the byte order (little or big endian), the length of the overall stream and the encoding of each one of the fields declared in the data model. Each field encoding has an associated *swe:Component* which only has two attributes: data type and ref. Data type points to an online resource defining its computer number format (i.e. unsigned byte, 32-bit integer, 64-bit floating point, etc.). The ref element links this component to a data model's field following the SensorML standard referencing rules.

F. SWE BRIDGE CONFIGURATION FOR NAXYS HYDROPHONE

The previous elements of the NAXYS SensorML description file are focused on properties related directly to the hydrophone. However, to setup an acquisition chain some additional information may be required, such as output format, recording time, duty cycle, etc. In the proposed architecture all the metadata is embedded within the hydrophone SensorML description, thus the configuration of the acquisition itself is also managed from the SensorML file.

The SWE Bridge includes a set of modules, each one targeting a specific task: process incoming data from the communication's interface, calculate sound pressure levels, generate WAV files, generate O&M files and access GPIO (general purpose input output) among others [19]. These modules are described using the SensorML standard and can be easily configured adding a specific section to a hydrophone SensorML description file. Using the *sml:typeOf* inheritance method, a SWE Bridge module invoked by referencing its identifier. With the *sml:Settings* element it is possible to change the parameters values. If not specified, the default value is used. The following XML excerpt shows how to configure the Sound Pressure Level module:

```
<sml:SimpleProcess gml:id="SoundPressureLevel" >
  <sml:typeOf
    xlink:title="swebridge:modules:soundPressureLevel"/>
  <sml:configuration>
    <sml:Settings>
      <sml:setValue ref="parameters/integrationTime">
        10</sml:setValue>
      <sml:setValue ref="parameters/frequencyBands">
        full 63 125 2000</sml:setValue>
      <sml:setValue ref="parameters/soundExposureLevel">
        true</sml:setValue>
      <sml:setValue ref="parameters/rootMeanSquare">
        true</sml:setValue>
      </sml:Settings>
    </sml:configuration>
</sml:SimpleProcess>
```

As its name indicates, the Sound Pressure Level module calculates SPL values. Each module has a set of parameters which can be set using the *sml:setvalue* elements. These parameters reflect non-trivial acquisition aspects that are not directly related to the sensor, but the user may need to adjust to fine-tune the acquisition. In the previous XML excerpt, the following parameters of the SPL algorithm have been set:

- **integration time**: Time window of the SPL measurements.
- **frequency bands**: Third-octave bands to calculated (*full* corresponds to the whole hydrophone's bandwidth)

- **sound exposure level**: Flag to determine if the SEL level should be calculated
- **root mean square**: Flag to determine if the root mean squared (RMS) level should be calculated

In order to generate a workflow within the SWE Bridge the modules need to be connected. The following XML excerpts shows how to connect two SWE Bridge modules:

```
<sml:connection>
  <sml:Link>
    <sml:source
      ref=".../HighFreqStream/.../dataOut"/>
    <sml:destination
       ref=".../SoundPressureLevel/.../dataIn"/>
  </sml:Link>
</sml:connection>
```

ACKNOWLEDGMENT

Researchers want to acknowledge the support of the Associated Unit Tecnoterra composed by members of Universitat Politècnica de Catalunya (UPC) and the Consejo Superior de Investigaciones Científicas (CSIC). This work used the EGI infrastructure with the dedicated support of INFN-CATANIA-STACK.

References

[1] J. Hildebrand, "Anthropogenic and natural sources of ambient noise in the ocean," *Marine Ecology Progress Series*, vol. 395, pp. 5–20, 2009.

[2] H. Slabbekoorn, N. Bouton, I. van Opzeeland, A. Coers, C. ten Cate, and A. N. Popper, "A noisy spring: the impact of globally rising underwater sound levels on fish," *Trends in Ecology & Evolution*, vol. 25, no. 7, pp. 419–427, 2010.

[3] R. Dekeling, M. Tasker, A. Van der Graaf, M. Ainslie, M. Andersson, M. André, J. Borsani, K. Brensing, M. Castellote, D. Cronin, J. Dalen, T. Folegot, R. Leaper, J. Pajala, P. Redman, S. Robinson, P. Sigray, G. Sutton, F. Thomsen, S. Werner, D. Wittekind, and J. Young, "Monitoring Guidance for Underwater Noise in European Seas, Part II: Monitoring Guidance Specifications," Publications Office of the European Union, Luxembourg, Tech. Rep., 2014, p. 49.

[4] A. J. Van der Graaf, M. A. Ainslie, M. André, K. Brensing, J. Dalen, R. P. A. Dekeling, S. Robinson, M. L. Tasker, F. Thomsen, and S. Werner, "European Marine Strategy Framework Directive Good Environmental Status (MSFD-GES): Report of the Technical Subgroup on Underwater noise and other forms of energy.," Publications Office of the European Union, Brussels, Belgium, Tech. Rep. February, 2012, p. 75.

[5] J. L. Miksis-Olds and S. M. Nichols, "Is low frequency ocean sound increasing globally?" *The Journal of the Acoustical Society of America*, vol. 139, no. 1, pp. 501–511, 2016.

[6] N. D. Merchant, K. L. Brookes, R. C. Faulkner, A. W. Bicknell, B. J. Godley, and M. J. Witt, "Underwater noise levels in UK waters," *Scientific Reports*, vol. 6, no. November, p. 36 942, 2016.

[7] M. van der Schaar, M. A. Ainslie, S. P. Robinson, M. K. Prior, and M. André, "Changes in 63Hz third-octave band sound levels over 42months recorded at four deep-ocean observatories," *Journal of Marine Systems*, vol. 130, pp. 4–11, 2014.

[8] M. Mustonen, A. Klauson, M. Andersson, D. Clorennec, T. Folegot, R. Koza, J. Pajala, L. Persson, J. Tegowski, J. Tougaard, M. Wahlberg, and P. Sigray, "Spatial and Temporal Variability of Ambient Underwater Sound in the Baltic Sea," *Scientific Reports*, vol. 9, no. 1, p. 13 237, 2019.

[9] G. Zhang, T. N. Forland, E. Johnsen, G. Pedersen, and H. Dong, "Measurements of underwater noise radiated by commercial ships at a cabled ocean observatory," *Marine Pollution Bulletin*, vol. 153, no. January, p. 110 948, 2020.

[10] S. Viola, R. Grammauta, V. Sciacca, G. Bellia, L. Beranzoli, G. Buscaino, F. Caruso, F. Chierici, G. Cuttone, A. D'Amico, V. De Luca, D. Embriaco, P. Favali, G. Giovanetti, G. Marinaro, S. Mazzola, F. Filiciotto, G. Pavan, C. Pellegrino, S. Pulvirenti, F. Simeone, F. Speziale, and G. Riccobene, "Continuous monitoring of noise levels in the Gulf of Catania (Ionian Sea). Study of correlation with ship traffic," *Marine Pollution Bulletin*, vol. 121, no. 1-2, pp. 97–103, 2017.

[11] H. R. Kolar, E. P. McKeown, M. E. Purcell, P. J. Gaughan, A. G. Westbrook, M. G. Barry, A. M. Akhriev, J. P. Hayes, A. Castelfranco, G. Nolan, D. J. Murray, K. P. Adlum, and D. F. Glynn, "The design and deployment of a real-time wide spectrum acoustic monitoring system for the ocean energy industry," in *2013 MTS/IEEE OCEANS - Bergen*, 2013, pp. 1–4.

[12] J. Tegowski, R. Koza, I. Pawliczka, K. Skóra, K. Trzcińska, and J. Zdroik, "Statistical, spectral and wavelet features of the ambient noise detected in the Southern Baltic sea," in *23rd International Congress on Sound and Vibration*, International Institute of Acoustics and Vibration (IIAV), 2016.

[13] L. Liu, L. Xiao, S. Q. Lan, T. T. Liu, and G. L. Song, "Using Petrel II Glider to Analyze Underwater Noise Spectrogram in the South China Sea," *Acoustics Australia*, vol. 46, pp. 151–158, 2018.

[14] A. Dassatti, M. van der Schaar, P. Guerrini, S. Zaugg, L. Houegnigan, A. Maguer, and M. Andre, "Onboard underwater glider real-time acoustic environment sensing," in *OCEANS 2011 IEEE - Spain*, IEEE, 2011, pp. 1–8.

[15] D. M. Toma, I. Masmitja, J. del Río, E. Martinez, C. Artero-Delgado, A. Casale, A. Figoli, D. Pinzani, P. Cervantes, P. Ruiz, S. Memè, and E. Delory, "Smart embedded passive acoustic devices for real-time hy-

droacoustic surveys," *Measurement*, vol. 125, pp. 592–605, 2018.

[16] J. Del Rio, D. M. Toma, T. C. O'Reilly, A. Broring, D. R. Dana, F. Bache, K. L. Headley, A. Manuel-Lazaro, and D. R. Edgington, "Standards-based Plug and Work for Instruments in Ocean Observing Systems," *IEEE Journal of Oceanic Engineering*, vol. 39, no. 3, pp. 430–443, 2014.

[17] E. Y. Song and K. Lee, "Understanding IEEE 1451 - Networked smart transducer interface standard - What is a smart transducer?" *IEEE Instrumentation and Measurement Magazine*, vol. 11, no. 2, pp. 11–17, 2008.

[18] A. Bröring, J. Echterhoff, S. Jirka, I. Simonis, T. Everding, C. Stasch, S. Liang, and R. Lemmens, "New generation Sensor Web Enablement," *Sensors*, vol. 11, no. 3, pp. 2652–2699, 2011.

[19] E. Martínez, D. M. Toma, S. Jirka, and J. D. Río, "Middleware for plug and play integration of heterogeneous sensor resources into the sensor web," *Sensors*, vol. 17, no. 12, p. 2923, 2017.

[20] A. Bröring, P. Maué, K. Janowicz, D. Nüst, and C. Malewski, "Semantically-Enabled Sensor Plug & Play for the Sensor Web," *Sensors*, vol. 11, no. 8, pp. 7568–7605, 2011.

[21] J. del Río, D. M. Toma, E. Martínez, T. C. O'Reilly, E. Delory, J. S. Pearlman, C. Waldmann, and S. Jirka, "A Sensor Web Architecture for Integrating Smart Oceanographic Sensors into the Semantic Sensor Web," *IEEE Journal of Oceanic Engineering*, vol. 43, pp. 830–842, 2017.

[22] U. K. Verfuß, M. Andersson, T. Folegot, J. Laanearu, R. Matuschek, J. Pajala, P. Sigray, J. Tegowski, and J. Tougaard, "Bias Standards for noise measurements. Background information, Guidelines and Quality Assurance," Tech. Rep., 2015.

[23] S. P. Robinson, P. A. Lepper, and R. A. Hazelwood, "Good Practice Guide for Underwater Noise Measurement," National Measurement Office, Marine Scotland, The Crown Estate, Teddington, England, Tech. Rep. 133, 2014.

[24] M. Botts and A. Robin, "OGC SensorML: Model and XML Encoding Standard 2.0," Open Geospatial Consortium, Wayland, MA, USA, Tech. Rep., 2014. [Online]. Available: http://www.opengis.net/doc/IS/SensorML/2.0.

[25] S. Cox, "Observations and Measuremets-XML Implementation," Open Geospatial Consortium, Wayland, MA, USA, Tech. Rep., 2011. [Online]. Available: http://www.opengis.net/doc/IS/OMXML/2.0.

[26] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J. W. Boiten, L. B. da Silva Santos, P. E. Bourne, J. Bouwman, A. J. Brookes, T. Clark, M. Crosas, I. Dillo, O. Dumon, S. Edmunds, C. T. Evelo, R. Finkers, A. Gonzalez-Beltran, A. J. Gray, P. Groth, C. Goble, J. S. Grethe, J. Heringa, P. A. t Hoen, R. Hooft, T. Kuhn, R. Kok, J. Kok, S. J. Lusher, M. E. Martone, A. Mons, A. L. Packer, B. Persson, P. Rocca-Serra, M. Roos, R. van Schaik, S. A. Sansone, E. Schultes, T. Sengstag, T. Slater, G. Strawn, M. A. Swertz, M. Thompson, J. Van Der Lei, E. Van Mulligen, J. Velterop, A. Waagmeester, P. Wittenburg, K. Wolstencroft, J. Zhao, and B. Mons, "The FAIR Guiding Principles for scientific data management and stewardship," *Scientific Data*, vol. 3, pp. 1–9, 2016.

[27] T. Tanhua, S. Pouliquen, J. Hausman, K. M. O'Brien, P. Bricher, T. de Bruin, J. J. Buck, E. F. Burger, T. Carval, K. S. Casey, S. Diggs, A. Giorgetti, H. Glaves, V. Harscoat, D. Kinkade, J. H. Muelbert, A. Novellino, B. G. Pfeil, P. Pulsifer, A. P. Van de Putte, E. Robinson, D. Shaap, A. Smirnov, N. Smith, D. P. Snowden, T. Spears, S. Stall, M. Tacoma, P. Thijsse, S. Tronstad, T. Vandenberghe, M. Wengren, L. Wyborn, and Z. Zhao, "Ocean FAIR data services," *Frontiers in Marine Science*, vol. 6, p. 440, 2019.

[28] A. Bröring, C. Stasch, and J. Echterhoff, "OGC Sensor Observation Service," Open Geospatial Consortium, Wayland, MA, USA, Tech. Rep., 2012, p. 163. [Online]. Available: http://www.opengis.net/doc/IS/SOS/2.0.

[29] S. Liang, C.-Y. Huang, and T. Khalafbeigi, *OGC SensorThings API Part 1: Sensing*, 2016. [Online]. Available: http://www.opengis.net/doc/is/sensorthings/1.0.

[30] R. A. Simons, *ERDDAP*, Monterey, CA, USA, 2019. [Online]. Available: ttps://coastwatch.pfeg.noaa.gov/erddap.

[31] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing*, 4th ed. Upper Saddle River, NJ, USA: Pearson Prentice Hall, 2007.

[32] F. Harris, "On the use of windows for harmonic analysis with the discrete Fourier transform," *Proceedings of the IEEE*, vol. 66, no. 1, pp. 51–83, 1978.

[33] G. Heinzel, A. Rüdiger, and R. Schilling, "Spectrum and Spectral Density Estimation by the Discrete Fourier Transform(DFT), Including a Comprehensive List of Window Functions and Some New At-Top Windows," Max-Planck-Institut fur Gravitationsphysik, Hannover, Germany, Tech. Rep., 2002.

[34] P. Stoica and R. Moses, *Spectral analysis of signals*. Upper Saddle River, NJ, USA: Pearson Prentice Hall, 2005.

[35] P. Welch, "The use of fast Fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms," *IEEE Transactions on Audio and Electroacoustics*, vol. 15, no. 2, pp. 70–73, 1967.

[36] K. Betke, T. Folegot, R. Matuschek, J. Pajala, L. Persson, J. Tegowski, and M. Wahlberg, "BIAS Standards for Signal Processing. Aims, Processes and Recommendations. Amended version," 2015.

[37] T. Paolo, F. Cristiano, O. Alessandro, and C. Paola, "Semantic Profiles for Easing SensorML Description: Review and Proposal," *ISPRS International Journal of Geo-Information*, vol. 8, p. 340, 2019.

[38] NERC, *The NERC vocabulary server: version 2.0*, 2016. [Online]. Available: http://vocab.nerc.ac.uk/ (visited on 11/16/2020).

[39] A. Kokkinaki, L. Darroch, J. Buck, and S. Jirka, "Semantically Enhancing SensorML with Controlled Vocabularies in the Marine Domain," in *Proceedings of the Geospatial Sensor Webs Conference*, 2016.

[40] S. Guan, H. Moustahfid, A. Milan, and J. Mize, *Metadata Conventions for IOOS Passive Acoustic Recordings (v1.0)*, 2013. [Online]. Available: https://mmisw.org/ont/ioos/passive_acoustic_metadata (visited on 07/17/2020).

[41] M. Nilsson, *ID3*, 2000. [Online]. Available: https://id3.org/.

[42] J. Del-Rio, M. Nogueras, D. M. Toma, E. Martinez, C. Artero-Delgado, I. Bghiel, M. Martinez, J. Cadena, A. Garcia-Benadi, D. Sarria, J. Aguzzi, I. Masmitja, M. Carandell, J. Olive, S. Gomariz, P. Santamaria, and A. Manuel Lazaro, "Obsea: A Decadal Balance for a Cabled Observatory Deployment," *IEEE Access*, vol. 8, pp. 33 163–33 177, 2020.

[43] A. Novellino, P. Gorringe, D. Schaap, S. Pouliquen, L. Rickards, and G. Manzella, "European marine observation data network - EMODnet Physics," in *2014 IEEE/OES Baltic International Symposium (BALTIC)*, IEEE, 2014, pp. 1–3.

[44] N. D. Merchant, P. Blondel, D. T. Dakin, and J. Dorocicz, "Averaging underwater noise levels for environmental assessment of shipping," *The Journal of the Acoustical Society of America*, vol. 132, no. 4, EL343–EL349, 2012.

[45] N. D. Merchant, T. R. Barton, P. M. Thompson, E. Pirotta, D. T. Dakin, and J. Dorocicz, "Spectral probability density as a tool for ambient noise analysis," *The Journal of the Acoustical Society of America*, vol. 133, no. 4, EL262–EL267, 2013.

[46] N. Lanteri, J. Legrand, B. Moreau, J. R. Lagadec, and J. F. Rolin, "The EGIM, a generic instrumental module to equip EMSO observatories," in *OCEANS 2017 - Aberdeen*, 2017, pp. 1–5.

[47] T. J. Rouphael, *RF and Digital Signal Processing for Software-Defined Radio*, 1st ed. Oxford, UK: Elsevier, 2009.

[48] A. Robin, "OGC SWE Common Data Model Encoding Standard," Open Geospatial Consortium, Wayland, MA, USA, Tech. Rep., 2011. [Online]. Available: http://www.opengis.net/doc/IS/SWE/2.0.

ENOC MARTÍNEZ received the B.S in telecomunications engineering and M.S. degree in electronic engineering from the Universitat Politècnica de Catalunya (UPC). He is currently pursuing the Ph.D. degree with the SARTI Research Group, Electronics Department, UPC. His fields of interest include interoperability, sensor web standards, underwater noise measurements, and underwater acquisition systems.

ALBERT GARCÍA-BENADÍ Received the bachelor's degree in physical sciences from the University of Barcelona, in 2000. He is currently the Technical and Quality Manager of the UPC Metrology and Calibration Laboratory accredited under ISO 17025 with accreditation number 152 / LC10.110. The current research interests are in acoustics measurement systems, atmospheric parametrization, meteorology radars, and oceanic environment.

DANIEL M. TOMA received the M.Sc. degree in electrical engineering from the Gheorghe Asachi Technical University, Iasi, Romania, in 2008, and the Ph.D. degree in the electronic engineering from the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain, in 2012. He is currently a member of the Remote Acquisition Systems and Data processing (SARTI), Electronic Engineering Department, UPC. His currentresearch interests include marine instrumentation, ocean observing systems, wireless *ad hoc* networks, interoperability in sensor networks, synchronization and scheduling in sensor networks, energy harvesting techniques, and passive acoustics.

ERIC DELORY (Dr.-Ing., Senior Member, IEEE) has worked for 25 years in research and development of monitoring systems, including signal processing, instrumentation and machine learning in biomedical applications and environmental monitoring. He joined the Oceanic Platform of the Canary Islands in 2010 as head of the observatory where his main activities have consisted in new systems developments and integration. He was the coordinator of NeXOS, a European collaborative project developing compact cost-effective sensors for the monitoring of ocean variables from autonomous platforms. He is involved in several European research infrastructure integration initiatives (EMSO, JERICO, GROOM), spanning fixed and mobile ocean observing platforms, to observe the ocean but also develop, test and validate new sensor technologies. He is Associate Editor for the IEEE Oceanic Engineering journal and Frontiers in Marine Science and has recently co-edited the book "Challenges and Innovations in Ocean In-Situ Sensors" (Elsevier, 2018).

SPARTACUS GOMÁRIZ (Member, IEEE) received the M.Sc. and Ph.D. degrees in telecommunication engineering from the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain. He is currently an Associate Professor with the Department of Electronic Engineering, UPC, and a member of the Remote Acquisition Systems and Data Processing (SARTI). His research interests include linear and nonlinear control theory, gain scheduled control, fuzzy control, design of navigation, guidance, and control systems for underwater vehicles. He is a member of IEEE Oceanic Engineering Society and Spanish Committee of Automation (CEA) of International Federation of Automatic Control (IFAC).

JOAQUÍN DEL-RÍO (Senior Member, IEEE) was born in Catalonia, Spain, in 1976. He received the B.Sc. and M.Sc. degrees in telecommunication engineering and electronic engineering from the Universitat Politecnica de Catalunya (UPC), Barcelona, Spain, in 1999 and 2002, respectively. Since 2001, he has been an Assistant Professor with the Electronic Engineering, Department, UPC. He is currently a member of the Remote Acquisition Systems and Data Processing (SARTI), focusing his research on electronic instrumentation, interoperability in marine sensor networks, and wireless sensor networks.

A.3 Article 3

Remote Configuration Service for Marine Observation Platforms through Sensor Web Enablement Components

---

Conference: *MTS/IEEE Oceans 2017 Aberdeen*
Publisher: IEEE
Date of Publication: 26 October 2017
DOI: OCEANSE.2017.8084792

Remote Configuration Service for Marine Observation Platforms through Sensor Web Enablement Components

Enoc Martínez,
Daniel M. Toma
and Joaquín del Río
SARTI Research Group,
Universitat Politècnica de Catalunya,
Vilanova i la Geltrú, Spain
Email: enoc.martinez@upc.edu

Simon Jirka
52 North Initiative for Geospatial
Open Source Software GmbH,
Münster, Germany

***Abstract*—In the past years important steps have been taken in order to improve both standardization and interoperability in marine acquisition systems, such as the adoption of the Open Geospatial Consortium's Sensor Web Enablement framework. Within this framework, standardized procedures to access the data from heterogeneous sensor resources have been proposed. However, due to the wide variety of software and hardware architectures of the different platforms, as well as power and communications constraints, the remote configuration of theses platforms still remains a challenge. In this work we present a standards-based approach to remotely configure sensors deployed in marine observation platforms through the Sensor Web Enablement framework.**

I. Introduction

Marine sensor systems and marine observation platforms present a vast variety of architectures and implementations depending on different factors, such as power constraints and available communication links. When integrating data from different sensor resources this variety can prove difficult. In order to address this issue, the ocean observing community has been progressively adopting the Open Geospatial Consortium's (OGC) Sensor Web Enablement framework (SWE) [1]. Its main objective is to abstract from the specific hardware and software layers of each sensor, providing a coherent and standardized framework to discover, access, visualize data and query sensors for new observations [2].

A. NeXOS Project

The EU funded NeXOS project aims to create innovative and interoperable sensors compliant with the SWE framework, avoiding proprietary protocols and non-standard architectures [3]. NeXOS sensors will be deployed in a wide variety of observation platforms such as gliders, buoys, ferries and cabled observatories among others. Each of this platforms has its own software and hardware layers, communication protocols, etc. Thus, a standardized mechanism to remotely manage and configure these sensors is required, regardless of the observation platform where they are deployed.

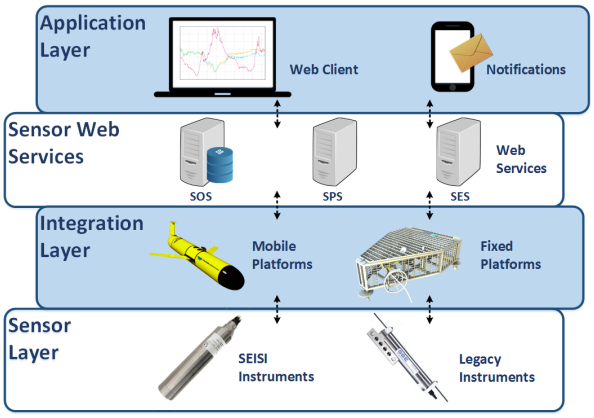


Fig. 1. Sensor Web Enablement Layer stack for ocean observing systems

Figure 1 shows the different layers of the SWE protocol stack for ocean observing systems. The Sensor Layer includes the sensor or sensor systems that are taking the measurements. The sensors can be legacy instruments or advanced instruments with smart interfaces such as the Smart Electronic Interface for Sensor Interoperability (detailed in section II-A). The Integration Layer includes the observation platforms hosting the sensors. These platforms are in charge of configuring the sensors, acquire data from them and send this data in a standardized format to the Sensor Web services. The Sensor Web Services layer forms the core of the NeXOS Sensor Web architecture: the services that store sensor data and metadata, services to query sensors for new measurements, services that provide notifications, etc. Finally the Application Layer contains front-end applications for the end users, such as a Web client to visualize data offered by the Sensor Web services.

978-1-5090-5278-3/17/$31.00 ©2017 IEEE

B. Sensor Web Enablement Standards and Protocols

The Sensor Observation Service (SOS) provides a standardized interface to register sensor resources, insert observations and store their metadata in a unified way [4]. The Sensor Model Language (SensorML) provides a well-defined and robust way of describing sensor resources as well as processing components associated to measurements [5]. It's main goal is to enhance interoperability, making the sensors description understandable by machines and shareable between intelligent Sensor Web nodes. The Observations and Measurements (O&M) standard specifies an abstract model as well as an XML encoding for observations and related data, such as features involved in the sampling process[6]. The PUCK protocol addresses identification, installation and configuration challenges faced by the sensors[7]. It defines a standard instrument protocol to store and automatically retrieve metadata and other information from the sensor device itself. Whereas the majority of SWE standards and protocols substitute non-standardized interfaces, this protocol is an add-on to sensor communication interfaces and can coexist with any previous communication protocol.

The Sensor Planning Service (SPS) is intended to provide a standardized interface to manage tasks that shall be executed by sensing devices [8]. Even though there are open source SPS implementation available, they were not yet fully suited to the specific requirements of marine applications. A specific challenge concerns the communication pattern: Running an SPS server directly on sensors or platforms would require a continuous connection to the network. This may not be a problem with Ethernet-enabled observation platforms such as cabled observatories. On the contrary this is a major drawback in platforms that do not have continuous broadband network interface. This is particularly critical in observation platforms deployed in remote and inaccessible regions, such as gliders, which only have very constrained satellite communications links during short periods of time. Thus, a communication mechanism was needed which allows platforms to actively query for new settings as soon as the platform is connected to a network. In this work we present a standards-based approach to remotely configure the acquisition process of marine observation platforms. This approach is discussed within the NeXOS project architecture.

II. NeXOS Architecture Components

The NeXOS architecture has been designed to integrate data from a wide variety of observation platforms. These platforms can be fixed or mobile and may present severe constraints on communications bandwidth and power consumption (i.e. a glider). Regardless of the platform's nature, all the data has to be acquired and processed in a standardized way. Figure 2 displays the different components of the architecture, as well as their interfaces and the layer where they belong.

A. Sensor Systems

The sensor systems used in this architecture can be legacy instruments with standard interfaces such as Ethernet or RS232

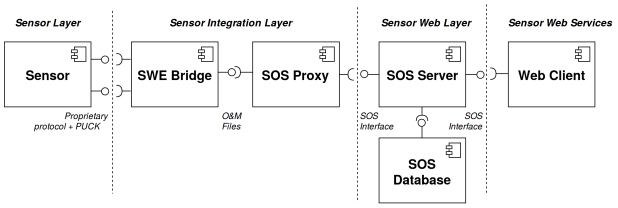


Fig. 2. NeXOS SWE Architecture components

using proprietary protocols. However, the sensors developed within NeXOS include the Smart Electronic Interface for Sensor Interoperability (SEISI). This interface is a software architecture envisioned to provide interoperable web access to marine sensors [9]. The SEISI architecture complies with standards defined by ISO, OGC and the INSPIRE directive, aimed to enable integration of marine sensors with existing observing systems. This interface relies on the PUCK protocol for auto-detection and auto-configuration capabilities. Within its PUCK memory, a SensorML description file is embedded.

B. Sensor Web Enablement Bridge

The Sensor Web Enablement Bridge (SWE Bridge) aims to bridge the gap between sensor systems and the Sensor Web. It is an auto-configurable, lightweight, cross-platform data acquisition software meant to be deployed in any kind of acquisition platform, fixed or mobile. It's main objective is to provide plug & play capabilities to any instrument, whether it is SWE-compliant or not. This software component configures itself according to the a SensorML file describing the interface of a sensor and automatically starts acquiring measurements from the sensor. In case of PUCK-enabled devices the SensorML file can be directly retrieved from the instrument itself. Alternatively, for legacy instruments the SensorML file should be stored on the observation platform.

C. SOS Proxy

The SOS Proxy is a software component which acts as an intermediary between the SWE Bridge and SOS servers. It injects the SWE Bridge output files containing observations into SOS servers. Additionally it is also used for the presented sensor configuration mechanism: The SOS proxy is capable of regularly checking for updated SensorML files, which may contain configuration updates to modify the current mission of the observation platform.

The communications between the SOS proxy and the SWE Bridge may vary depending on the nature of the platform. If the observation platform does not have communications constraints (i.e. a cabled underwater observatory) they can be deployed on the same physical platform.

On the contrary if the platform presents communications and/or power constraints, the platform operator may decide to deploy the SWE Bridge on the observation platform and the SOS Proxy on a shore station server, as shown in Figure 3. In this case, the operator is in charge to provide a channel to transmit the generated files through the platform's

communications interface to the shore station server. This is the typical scenario for satellite-controlled platforms, such as gliders and unmanned surface vehicles. Nonetheless, the proprietary communication channel is transparent and does not affect the rest of the architecture.

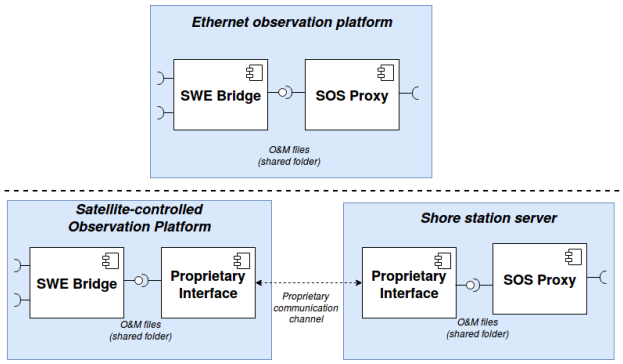


Fig. 3. Integration Layer in Ethernet-enabled observation platforms (top) and satellite-controlled observation platforms (bottom)

D. SOS Server

The Sensor Observation Service acts as a server for storing and managing both sensor metadata and sensor data [4]. In case of the NeXOS project this SOS server consists of the 52North open source SOS implementation running on a PostgreSQL database. For ingesting metadata about a sensor into the SOS server (e.g. registering a new sensor) the transactional *InsertSensor* operation of the SOS interface is used. The upload of data to the SOS server is achieved through the so called *ResultHandling* operations of the SOS standard (*InsertResultTemplate* and *InsertResult*) to ensure an efficient data transfer. Using a web client it is possible to visualize both sensor data and metadata stored in the SOS database.

Complementary to the data access functionality, the SOS server also offers a dedicated operation to modify the metadata of a sensor by uploading changed SensorML files (e.g. with sensor settings that were changed). Within the NeXOS project. To enable such an editing of sensor settings, the NeXOS Sensor Web was enhanced to make it also possible to modify the existing SensorML description files for specific instruments (e.g. modifying acquisition parameters such as sampling rate, post-measurement processing, etc).

III. Remote Configuration Service

The Remote Configuration Service is a SOS-based approach to modify the sensor's configuration in near-real time. To enable the configuration workflow an updated SensorML file containing changed configuration parameters has to be delivered through the whole chain, from the SOS server the sensor. This file contains the new acquisition configuration parameters, which the SWE Bridge will decode and use in the acquisition process. Additionally these new description file will be written into the sensor's PUCK payload. Thus the full chain, from the sensor to the SOS Server, will be aware of the current configuration described within the SensorML file. To determine if there is a new configuration file the SOS Proxy periodically sends requests to the SOS Server. If a new configuration file is available it is downloaded and sent to the SWE Bridge, which will configured accordingly.

A. Extended SOS

Following the SOS standard, before starting to inject data collected by a sensor to a SOS server it is necessary to register this new sensor with the *InsertSensor* operation. Afterwards, the data structure of the O&M data containing sensor observations needs to be registered using the *InsertResultTemplate* operation. The generation of the transactional files used by the sensor's registration procedure may prove difficult as in-deep knowledge of the SensorML and SOS standards is required. In the presented architecture default SOS functionality is extended by adding the option to register a new sensor by simply transferring its SensorML description to an SOS server. In this case, additional, specific parameters of the *InsertSensor* operation are not required. Instead a more lightweight approach requiring less complex business logic on the instrument/platform side is achieved.

B. SensorML Configuration File

The sensor's SensorML description file is a key piece of the architecture, as it contains sensor metadata, the sensor's data acquisition parameters and the necessary parameters to register the sensor to the SOS server. The SensorML description file for the sensor needs to include information concerning different aspects of the acquisition chain:

- **Sensor Information**: Information related to the sensor's deployment and the sensor itself such as uid, instrument commands, communications interface, platform where it is installed, location, deployment history, etc.
- **Acquisition Information**: Specific information for the SWE Bridge, defining the sampling rate, which parameters need to be stored in the resulting O&M files, the encoding required by the resulting files, etc.
- **SOS-related Information**: Information required by the SOS server to register the sensor. This includes *ObservableProperties*, *featuresOfInterest*, *offering*, *uniqueID* and *capabilities*.

C. Remote Configuration Procedure

When a PUCK enabled sensor is connected to an acquisition platform, the SWE Bridge detects it automatically and retrieves its internal SensorML file as shown in Figure 4. After configuring itself with the information contained in the SensorML file, the SWE Bridge passes the SensorML file to the SOS Proxy, and the SOS Proxy sends it to the SOS Server (*InsertSensor* operation). The SOS Proxy also decodes the SensorML file and generates an *insertResultTemplate*, which is also sent to the server (*InsertResultTemplate* operation).

Once the sensor has been registrated, the SWE Bridge starts to gather sensor data periodically, according to the SensorML description (as shown in Figure 5). The gathered data is stored

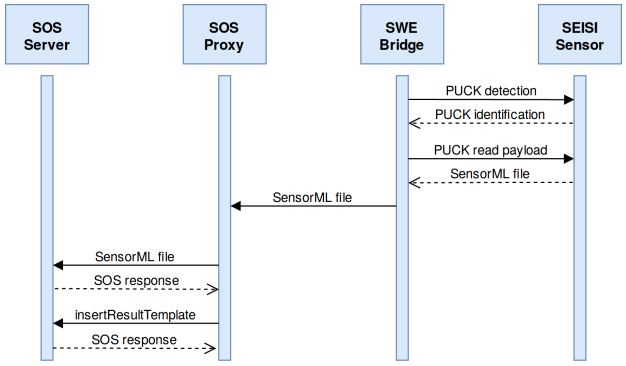


Fig. 4. Registration of a new sensor to the SOS server

in an O&M structure and transferred to the SOS server relying on the *InsertResult* operation.

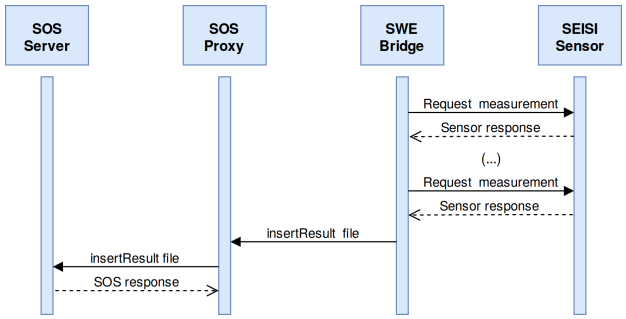


Fig. 5. SWE Bridge data acquisition and O&M file generation

After some time it may be possible that the user responsible for the sensor may want to change the sensor configuration. The configuration can be modified by updating the SensorML description file and sending it back to the sensor using a Web client.

The first step is to identify the sensor, so a *GetCapabilities* request is sent to the SOS server (Figure 6). The server response includes metadata regarding the SOS server, including registered sensors. After the Web client decodes the response, it is aware of all the registered sensors, and they can be selected by the user. The next step is to access the last description of the sensor of interest using a *DescribeSensor* request. Once the last version of the file is received, it can be modified using a SensorML editor, i.e. 52 North's smle.

Once the user has modified the description to fit the new sensor configuration, the file can be uploaded using the *UpdateSensorDescription* operation. For security issues *UpdateSensorDescription* operation can only be performed after a successful authentication.

The SOS Proxy periodically sends a *DescribeSensor* request to the SOS server, whose response is always contains the latest version of the SensorML file used to register the sensor. Then SOS proxy has to determine whether the file has been updated or not. If it has been updated, it means that a new configuration for the SWE Bridge is available. Thus the updated file is sent

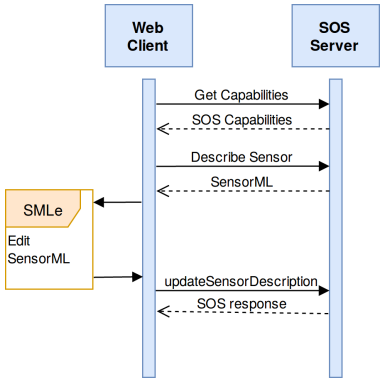


Fig. 6. Updating a SensorML description file using a Web Client with the SMLe editor

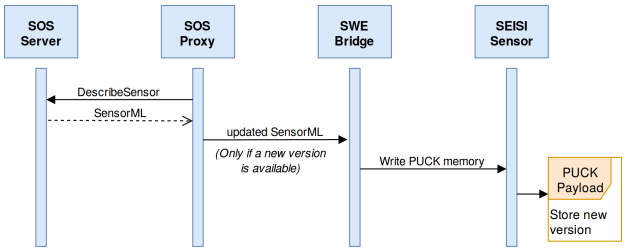


Fig. 7. Configuration of a sensor deployed in an acquisition platform using the remote configuration service

to the SWE Brige, who reconfigures itself. The final step, is to update the SensorML file within the sensor's PUCK payload. Once the sensor is recovered after the mission, it will always contain the last version of the SensorML descriptions.

IV. Conclusion

The presented Remote Configuration Service provides an universal approach to deliver new configuration and new missions to heterogeneous sensor resources. Its SOS-based approach permits that sensor deployed in platforms with severe power and communications constraints can be effectively configured from a Web client.

A drawback of this approach is the use of an extension of the SOS server. This may prove an issue when using SOS servers outside this architecture.

For the future it is planned to investigate how this approach may also be mapped to the SPS interface. In this case, a SPS server may act as an intermediary server which collects configuration change requests that can be retrieved by platforms as soon as they have the necessary connectivity. Furthermore research might comprise the integration of IoT protocols such as MQTT for delivering configuration changes to devices.

Acknowledgment

NeXOS is a collaborative project funded by the European Commission 7th Framework Programme, under the call OCEAN-2013.2 - The Ocean of Tomorrow 2013 - Innovative multifunctional sensors for in-situ monitoring of marine environment and related maritime activities (grant agreement No

614102). It is composed of 21 partners including SMEs, companies and scientific organizations from 6 European countries.

REFERENCES

[1] M. Botts, G. Percivall, C. Reed, and J. Davidson, "OGC (R) Sensor Web Enablement: Overview and High Level Architecture," *Lecture Notes In Computer Science*, vol. 4540, no. December, pp. 175–190, 2007.

[2] A. Bröring, J. Echterhoff, S. Jirka, I. Simonis, T. Everding, C. Stasch, S. Liang, and R. Lemmens, "New generation Sensor Web Enablement," vol. 11, no. 3, pp. 2652–2699, 2011.

[3] E. Delory, A. Castro, C. Waldmann, J. F. Rolin, P. Woerther, J. Gille, J. Del Rio, O. Zielinski, L. Golmen, N. R. Hareide, J. Pearlman, and R. Garello, "Objectives of the NeXOS project in developing next generation ocean sensor systems for a more cost-efficient assessment of ocean waters and ecosystems, and fisheries management," *Oceans 2014 - Taipei*, 2014.

[4] *OGC®Sensor Observation Service Interface Standard 2.0*, Open Geospatial Consortium Std. OGC 12-006, 2012.

[5] *OGC®SensorML: Model and XML Encoding Standard 2.0*, Open Geospatial Consortium Std. OGC 12-000, 2014.

[6] *OGC®Observations and Measurements - XML Implementation 2.0*, Open Geospatial Consortium Std. OGC 10-025r1, 2011.

[7] *OGC®PUCK Protocol Standard Version 1.4*, Open Geospatial Consortium Std. OGC 09-127r2, 2012.

[8] *OGC®Sensor Planning Service Implementation Standard 2.0*, Open Geospatial Consortium Std. OGC 09-000, 2011.

[9] D. M. Toma, J. Del Rio, S. Jirka, E. Delory, J. Pearlman, and C. Waldmann, "NeXOS smart electronic interface for sensor interoperability," *MTS/IEEE OCEANS 2015 - Genova: Discovering Sustainable Ocean Energy for a New World*, 2015.

A.4 Article 4

Seamless Integration of EMSO Generic Instrument Module into the Internet Using Sensor Web Components based on OGC SWE framework

---

Conference: *MTS/IEEE Oceans 2017 Aberdeen*
Publisher: IEEE
Date of Publication: 26 October 2017
DOI: 10.1109/OCEANSE.2017.8084676

Seamless Integration of EMSO Generic Instrument Module into the Internet Using Sensor Web Components based on OGC SWE framework

Daniel. M. Toma, E. Martínez, J. Del Rio
SARTI Research Group. Electronics Department, Universitat Politècnica de Catalunya. Vilanova i la Geltrú, Spain

I. Bghiel, Ó. Garcia, J. Dañobeitia
Marine Technology Unit, UTM-CSIC. Barcelona, Spain

N. Lantéri
French Research Institute for Exploitation of the Sea, IFREMER Centre Méditerranée, La Seyne sur Mer, France

***Abstract*— The EMSODEV [1] (European Multidisciplinary Seafloor and water□column Observatory DEVelopment) is a UE project whose general objective is to set up the full implementation and operation of the EMSO distributed Research Infrastructure (RI), through the development, testing and deployment of an EMSO Generic Instrument Module (EGIM). The scientific drivers for developing and deploying the EGIM across a set of observatories in European Seas are manifold, spanning requirements to collect observations for understanding climate change, marine ecosystems, and geo-hazard early warning research. The EGIM (EMSO Generic Instrument Module) is designed to consistently and continuously measure parameters of interest for most major science areas covered by EMSO. This research infrastructure provides accurate records on marine environmental changes from distributed regional nodes around Europe. EGIM is able to operate on any EMSO node, mooring line, sea bed station, cabled or non-cabled and surface buoy. In fact, a central function of EGIM within the EMSO infrastructure is to have a number of ocean locations where the same set of core variables are measured homogeneously: using the same hardware, same sensor references, same qualification methods, same calibration methods, same data format and access, same maintenance procedures. our contribution to the implementation of the EGIM data acquisition system module focusses on the development of a generic software for sensor web enablement. Through this generic software, the EGIM status data is directly inserted into a centralised SOS (Sensor Observation Service) server and into a laboratory monitor system (Zabbix LabMonitor) for recording events and alarms.**

Keywords*— *EMSO, EGIM, Ocean Observatories, OGC SOS, OGC SWE

I. INTRODUCTION

The scientific drivers for developing and deploying the EGIM across a set of observatories in European Seas are manifold, spanning requirements to collect observations for understanding climate change, marine ecosystems, and geo-hazard early warning research. As illustrated in Figure 1, the EGIM will utilize a comprehensive set of sensors and devices that meet particular technology readiness thresholds to collect observations including temperature, pressure, salinity, dissolved oxygen, turbidity, chlorophyll fluorescence, currents, and passive acoustics.

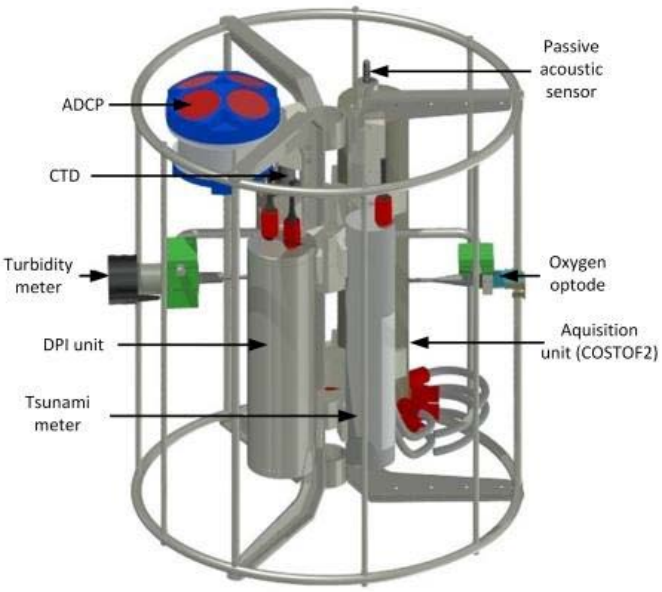


Figure 1 EGIM prototype components

978-1-5090-5278-3/17/$31.00 ©2017 IEEE

Relatively novel sensors will also be considered including those for pH, pCO2, and nutrients. Overall, this system will address the fullest possible set of Essential Climate Variables (e.g. from the GCOS-Global Climate Observing System programme of the WMO; www.wmo.int) at EMSO nodes [1].

The system can deliver data that can support the Global Ocean Observing System –Essential Ocean Variables concept, as well as the Marine Strategy Framework Directive towards evaluating environmental status. The EGIM is able to operate on any EMSO node type: mooring line, sea bed station, cabled or non-cabled and surface buoy to monitor environmental parameters over a wide depth range. Operating modes, power requirements, mechanical design can adapt to the various EMSO node configurations. Moreover, the EGIM will provide unprecedented support for full standardization across EMSO, key for understanding regional scale phenomena. Data will be made coherent and attractive for the modelling community and for other potential stakeholders as shown in Table 1.

Table 1 Core variables captured by the EGIM - EMSO Generic Instrument Module, and their cross-disciplinary application

Variable	Geosciences	Physical Oceanography	Biogeochemistry	Marine Ecology
Temperature	X	X	X	X
Conductivity	X	X	X	X
Pressure	X	X	X	X
Dissolved O_2	X	X	X	X
Turbidity	X	X	X	X
Ocean currents	X	X	X	X
Passive acoustics	X			X

Within EMSO, the EGIM aims to have a number of ocean locations where the same set of core variables are measured homogeneously. This is achieved through a generic acquisition software for sensor web enablement. The generic software for sensor web enablement together with the SOS server is located, in the EMSO Cyberinfrastructure (CI), between the data source (EGIM) and the data management system. The generic software for sensor web enablement has two main functionalities, first to guarantee that the data is recorded properly from the EGIM hardware and second to register and insert the recorded data into a standardize OGC SOS server that works as a gateway for the EMSO data management system.

II. The SWE Framework

The OGC Sensor Web Enablement (SWE) working group defines standards for sensor data and sensor services. Accordingly, to Botts et al. [2] "a Sensor Web refers to web accessible sensor networks and archived sensor data that can be discovered and accessed using standard protocols and application program interfaces (APIs)".

To achieve the vision of the Sensor Web, the OGC SWE initiative defines standards for encoding of sensor data as well as standards for service interfaces to access sensor data, task sensors or send and receive alerts. We have identified several categories of data shared between EGIM and CI such as: component descriptive data, command data, instrument data, engineering data, metadata.

A. Sensor Model Language

To provide the description of all these categories of data we use the SensorML 2.0 standard. SensorML supports the ability to describe the components and encoding of real-time data streams, and to provide a link to the data stream itself. Based on the SensorML description of each EGIM component, the generic software for sensor web enablement can automatically connect to a real-time data stream, parse the data stream and generate transaction compliant with Observation & Measurement standard 2.0 which can be directly injected in the OGC SOS server. Additional metadata like quality, calibration information or technical attributes can also be nested in SensorML descriptions.

B. Observations & Measurements

The Observations & Measurements (O&M) specification defines basic models and encodings for observations and measurements made by sensors [3]. The basic observation model contains five components:

i. Procedure element points to the procedure (usually a sensor/instrument deployed within EGIM), which produced the value of the observation.

ii. Phenomenon that was observed is referenced by the observedProperty element.

iii. FeatureOfInterest refers to the real world object to which the observation belongs. The referenced feature also contains the location information of the observation.

iv. SamplingTime attribute indicates the time, when the observation was sampled.

v. Observation value itself is contained in the result element.

C. *Sensor Observation Service*

The Sensor Observation Service (SOS) provides a standardized web service interface which allows clients to access descriptions of associated sensors and their collected observations. To accomplish the SOS Gateway requirement, and once we have analysed the actual state of the art on SOS implementations, we have decided to use the 52 North SOS 2.0 implementation [4]. This 52 North SOS 2.0 has capabilities to aggregate readings from live, in-situ and remote sensors. The service provides an interface to make sensors and sensor data archives both accessible through an interoperable web-based interface, using SensorML and Observation and Measurements (O&M). The main SOS 2.0 interfaces offered with this implementation are:

i. Core Extension

- GetCapabilities, for requesting a self-description of the service
- GetObservation, for requesting the pure sensor data encoded in Observations & Measurements 2.0 (O&M)
- DescribeSensor for requesting information about a certain sensor, encoded in a Sensor Model Language 1.0.1 (SensorML) instance document.

ii. *Enhanced Extension*

- GetFeatureOfInterest, for requesting the GML 3.2.1 encoded representation of the feature that is the target of the observation.
- GetObservaitonById, for requesting the pure sensor data for a specific observation identifier

iii. Transactional Extension

- InsertSensor, for publishing new sensors
- UpdateSensorDescription, for updating the description of a sensor
- DeleteSensor, for deleting a sensor
- InsertObservation, for publishing observations for registered sensors

iv. Result Handling Extension

- InsertResultTemplate, for inserting a result template into a SOS server that describes the structure of the values of an InsertResult of GetResult request.
- InsertResult, for uploading raw values accordingly to the structure and encoding defined in the InsertResultTemplate request
- GetResultTemplate, for getting the result structure and encoding for specific parameter constellations
- GetResult, for getting the raw data for specific parameter constellations.

For attending client requests, we have deployed the Helgoland web client application from 52North [5] [6], to visualize the real time and historical data using the SOS gateway as illustrated in Figure 2.

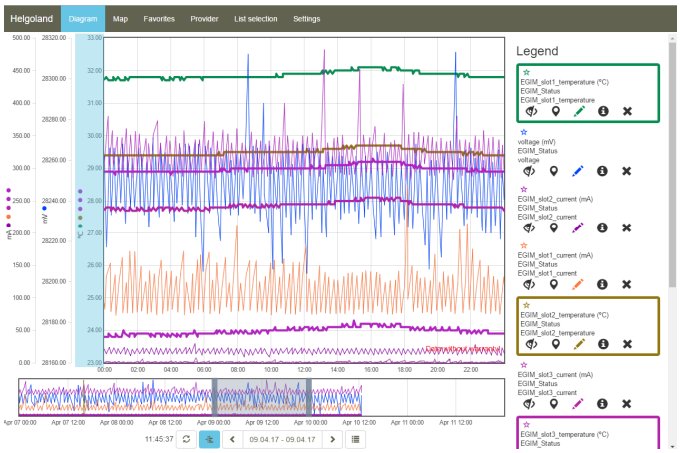


Figure 2 Visualization of EGIM status data on 52North Helgoland SOS Client.

III. Application of SWE in the EMSODEV Project

Within EMSODEV, the efficiency of the in-situ data processing chain is improved by enabling end-users to access multi-domain sensor information. The in-situ sensors are connected via an intelligent and versatile network infrastructure. Two use cases demonstrate the effectiveness of EMSODEV: EGIM deployed in cabled mode in underwater cabled observatories and EGIM deployed in autonomous mode as fixed point observatory. These experiments are described in the following sections.

A. *EGIM deployed in cabled mode in underwater cabled observatories*

This scenario of the EMSODEV project is centered on the deployment of the EGIM module as a node for the EMSO cabled observatories such as the ANTARES in Ligurian Sea [7], the OBSEA in Balearic Sea [8] or the EMSO-MOLÈNE in Molène Island [9]. These cable observatories support instruments deployed on, below, and/or above the seabed, continuously acquiring oceanographic and geophysical time series. Two-way real-time data delivery and control through electro-optic connectors and cables extend from the seafloor to land. The EGIM module deployed on these observatories contains a generic package for measuring key chemical and physical parameters (for example the temperature, salinity, concentration of oxygen, turbidity, passive acoustics, pressure/depth, and currents). Moreover, these observatories may also be used as points for the installation of specific

experiments defined by the user, for specific applications of interest [13].

A practical demonstration of this scenario was conducted in the Balearic Sea in the OBSEA Observatory. Overall, this scenario includes the generic EGIM package with status sensors and six different kinds of scientific instruments. Firstly, the EGIM platform is equipped with a set of internal sensors for measuring key status parameters (for example, internal temperature, pressure, current, voltage, and water intrusion). This enables the control center to gain a general overview of the behavior of EGIM module and to identify the power consumption of the scientific instruments and their evolution over time. The images captured by the EGIM control module are ingested by a SOS instance, through generic acquisition software, which allows the standardized retrieval of these parameters. Secondly, scientific instruments (for example, Seabird SBD37-CTD, Seabird SBE54-P tsunami, Aandeera 4330 – O2, Wetlabs NTU–Fluo, RDI Workhorse-ADCP, and icListen-hydrophone) were used for measuring key chemical and physical parameters.

The EGIM instruments sends the sensor data to the control center where the generic acquisition software is producing automatically O&M queries. Once the acquisition agent produces the O&M queries, a 'proxy SOS' tool is used to insert automatically all the data into the SOS instance. Hence, the generic acquisition software is registering any new sensors connected to EGIM and send the O&M queries to the SOS instance for each new data acquired from EGIM instruments. Moreover, the acquisition agent is generating also JSON requests to Zabbix server [14], in order to add these values to the database of the Lab Monitor.

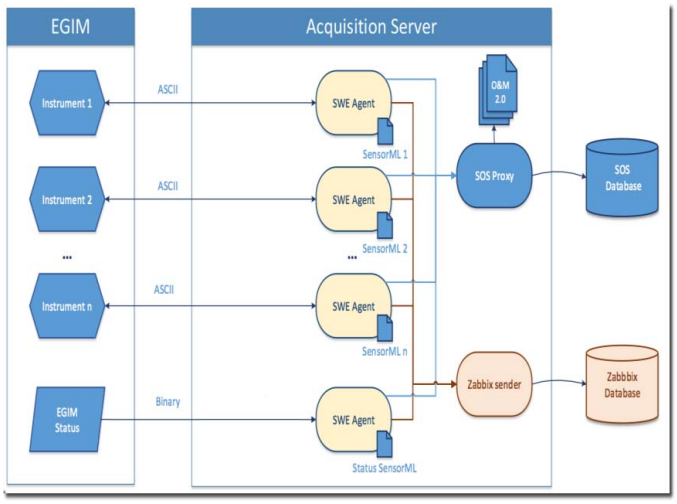


Figure 3 Overview of the EGIM Acquisition Components

We have identified several categories of data shared between EGIM and CI. The following define each one:

- Component descriptive data – Description of the platform/instrument configuration including instrument types, serial numbers, position of the deployment, calibration parameters.
- Command data – Commands and associated attributes such as when a command is scheduled to be executed.
- Instrument data – Data produced by the platform instruments, associated time tags, and attributes identifying the specific source instrument.
- Engineering data – Data describing the operational status of the system components.
- Metadata – Data describing the data. Metadata are data describing a resource like an instrument or an information resource.

To provide the description of all these categories of data we use the SensorML 2.0 standard. SensorML supports the ability to describe the components and encoding of real-time data streams, and to provide a link to the data stream itself [6]. Thus allows, one connecting directly to a real-time data stream directly from a SensorML description, and use a generic data reader to parse the data stream. Describing a data stream into or out of a process (or sensor/actuator) is accomplished by having the input or output be of type DataInterface. The DataInterface element allows describe the DataStream, as well as provides for an optional interface description.

The acquisition agent (SWE Agent reads and decodes this file, encoded in EXI format. With the decoded information it autoconfigures itself, opening a communication port with the EGIM-deployed instrument through an Ethernet connection, with the capability to use both TCP and UDP protocols. Then starts getting information from the instrument in push or pull mode. The data retrieved from the instrument is stored in XML files, following the insertResult format. This format is compliant with the Observation & Measurement standard 2.0 and can be directly injected in the SOS database.

B. EGIM deployed in autonomous mode as fixed point observatory

This scenario of the EMSODEV project is centered on the deployment of the EGIM module that works autonomously via satellite for EMSO observatories such as the PAP in Porcupine Abyssal Plain [10], the EMSO-Azores in Azores Islands [11] or the MARDEP in Marmara Sea [12]. These EMSO stand-alone observatories are autonomous installations of instruments, sensors and command modules operating in the long-term on and beneath the seafloor, in the water column and/or at the sea surface. They support the operation of a number of instrument packages related to various disciplines and scientific questions. They are characterized by a stand-alone configuration for power and data, and a limited capacity

of connection from the surface, including capability of bidirectional communication, transfer status parameters and a limited quantity of real-time or near real-time data [13].

As illustrated in Figure 4 an on-board communication system sends the sensor data to the control center through the satellite link where the generic acquisition software is producing automatically O&M queries. The generic acquisition software is registering any new sensors connected to EGIM and send the O&M queries to the SOS instance for each new data acquired from EGIM instruments. Moreover, if the EGIM internal memory is recovered at the end of deployment, the full data are ingested by a SOS instance, through the generic acquisition software, which allows the standardized retrieval of these parameters.

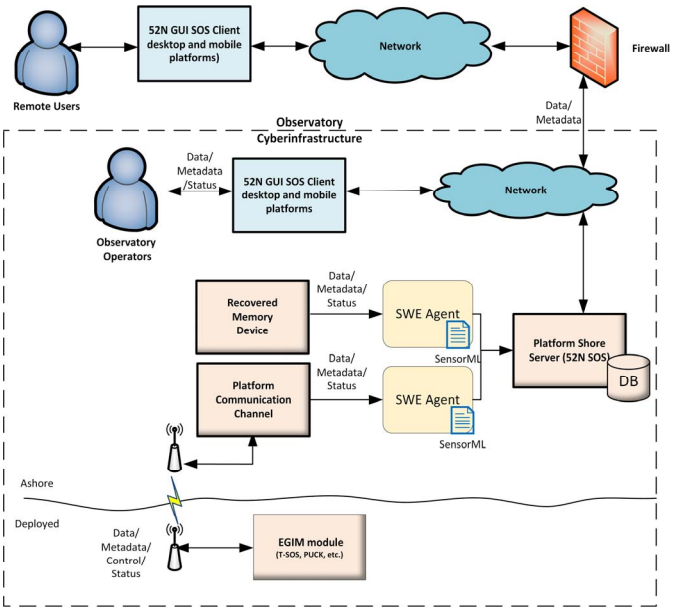


Figure 4 Application of SWE in EMSO stand-alone observatories

IV. Experiences and Outlook

The implementation of the applications described within this article leads to the experience that the OGC SWE architecture has now reached a solid and mature state. Especially, the standardized interfaces of SOS the encodings O&M and SensorML provide a sound foundation for building web based applications on top of sensors and sensor networks as the ones envisioned in the EMSODEV project.

During the realization of the different systems described in this article, the often generic character of the OGC specifications was very challenging. The definition of profiles, describing subsets of OGC service interfaces and data formats, that are adapted to specific domains like the ocean observatories, would significantly facilitate their practical application.

Finally, a further work item is to improve the system complementing the way that Data Management Platform (DMP) should get the data. Initially, this configuration requires a connection polling to request data from DMP to SOS Server. This implies two operations for each request data. We have installed the Sensor Event Service in SOS server, with the objective that, users with a publish/subscribe-based interface, could access to sensor data and measurements located at SOS server. Basically, SES produces notifications and provides methods to subscribe for notifications and retrieve the latest notification. Meanwhile, users can also register new sensors dynamically and send notifications to the service.

Acknowledgment

This study benefited from H2020 INFRADEV□3□2015 EMSODEV Project n°676555.

References

[1] EMSO project site http://www.emso-eu.org/site/projects.html

[2] OGC® SensorML: Model and XML Encoding Standard, Version 2.0.0, OGC 12-000, Wayland, MA, USA: OGC.

[3] OGC Abstract Specification Geographic information — Observations and measurements, Version 2.0, OGC and ISO 19156:2011(E), Wayland, MA, USA: OGC

[4] Arne Bröring, Christoph Stasch, Johannes Echterhoff "OGC® Sensor Observation Service Interface Standard", http://www.opengis.net/doc/IS/SOS/2.0 , 2012-04-20

[5] 52 North SOS 2.0 implementation, http://52north.org/communities/sensorweb/sos/

[6] https://github.com/52North/helgoland

[7] http://www.emso-fr.org/EMSO-Ligure-Nice/Scientific-objectives

[8] http://www.upc.edu/cdsarti/OBSEA

[9] http://www.emso-fr.org/fr/EMSO-Molene

[10] http://noc.ac.uk/pap

[11] http://www.emso-fr.org/fr/EMSO-Azores

[12] http://www.esonet.marmara-dm.itu.edu.tr

[13] http://www.emso-eu.org/site/ocean-observatories.html

[14] Zabbix Monitoring System. http://www.zabbix.com/product.php

[15] del Río, J.; Mihai Toma, D.; O'Reilly, T.C.; Bröring, A; Dana, D.R.; Bache, F.; Headley, K.L.; Manuel-Lazaro, A; Edgington, D.R., "Standards-Based Plug & Work for Instruments in Ocean Observing Systems," Oceanic Engineering, IEEE Journal of , vol.39, no.3, pp.430,443, July 2014 doi: 10.1109/JOE.2013.227327

Appendix B

SWE Bridge User Manual

SWE Bridge

User Manual

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Document Version: 0.2
Date: February 10, 2021
Author: Enoc Martínez
Contact: enoc.martinez@upc.edu

Revision History

Revision	Date	Author(s)	Description
0.1	12.12.18	E. Martínez	First draft
0.11	08.05.19	E. Martínez	Added new features on v0.71
0.2	30.01.21	E. Martínez	Added description, examples and features in v0.9

Contents

1 Introduction

The SWE Bridge is a universal driver that aims to interface any scientific instrument. It relies on instrument descriptions in SensorML language to abstract their characteristics, communications protocols, etc. Using this sensor abstractions, it is able to interface almost any sensor. Its hardware-independent architecture facilitates its deployment in virtually any observation platform (dataloggers, underwater gliders, unmanned surface crafts, etc.).

The SWE Bridge implements a set of modules, each one targeting a specific sensor-related operation. These modules can be interconnected, creating complex acquisition dataflows. Some operations implemented using modules are: send/receive commands, generate data files (O&M, CSV, etc.), apply calibration, subsample incoming data, power on/off instruments and many more.

It implements the OGC PUCK protocol, so any PUCK-enabled instrument can be seamlessly integrated with the SWE Bridge in a plug & play manner.

1.1 Standards and Protocols

The SWE Bridge builds upon numerous existing standards rather than reinventing the wheel, specially those conforming the Sensor Web Enablement (SWE) framework and the OGC PUCK Protocol. Some of these standards are:

1.1.1 OGC PUCK Protocol

The OGC PUCK protocol is an add-on that can be implemented in any serial or Ethernet sensor alongside with any proprietary protocol, rather than replacing it [1]. This protocol defines a set of commands that grant transparent access to an internal memory, named *OGC PUCK payload*. This payload is frequently used to store sensor metadata. Another key feature of the OGC Puck protocol is its *softbreak* operation, which provides on-the-fly detection without any prior knowledge of the sensor. In this case, the PUCK Payload is oftenly used to embed a Sensor Deployment File (see section 3).

1.1.2 Sensor Model Language

The OGC Sensor Model Language (SensorML) encodes detailed sensor descriptions within XML files [2]. Its main goal is to enhance interoperability, making sensor descriptions understandable by machines and shareable between intelligent nodes. Moreover, additional information related to specific deployments can also be encoded using this standard. Thus, both sensor configuration and measurement operations can be addressed with the SensorML standard.

It is highly flexible and modular as it can describe almost every property related to a sensor or sensor-related process. However this flexibility and modularity can prove a double-edged sword, as the same information can be encoded in different ways, increasing the difficulty to generate smart processes capable of interpreting SensorML definitions. This document aims to provide a guide to generate SensorML documents compatible with the SWE Bridge.

1.1.3 SWE Common Data Model

The primary focus of the SWE Common Data Model is to define and package sensor related data in a self-describing and semantically enabled way [3]. The main objective is to achieve interoperability, so that sensor data can be better understood by machines, processed automatically in complex workflows and easily shared. SensorML files extensively uses this standard to model sensor data streams.

1.1.4 Sensor Observation Service

The Sensor Observation Service (SOS) standard provides the set of operations required to provide access sensor observation data/metadata as well as to register and archive sensor data and metadata within a data repository [4]. Due to its role as a data and metadata archive, this standard is a key piece of any Sensor Web infrastructure.

1.1.5 Observations & Measurements

The Observations and Measurements (O&M) standard specifies an abstract model as well as XML encoding for observations and related data, such as features involved in the sampling process [5]. It provides a uniform and unambiguous way to encode sensor measurements.

1.1.6 Efficient XML Interchange

The Efficient XML Interchange (EXI) is a World Wide Web Consortium's (W3C) format that enables the compression of large ASCII XML files into efficient binary files, significantly reducing its size. This format has been designed to allow information sharing between devices with constrained resources [6].

1.2 SWE Common Data Types

The SWE Bridge has been designed to be compatible with the SWE Common Data Model standards. Thus, all parameters and data components use the SWE

Common Data Model encoding. Table 1.2 provides an overview on the used data types, their C equivalents and their lengths in bytes.

need to be configured using the SWE Common data types.

SWE Common	**C language**	**Length**
Quantity	double	64 bit
Count	int	32 or 64 bit
Boolean	unsigned char	8 bit
Text	char*	(variable)
Category[1]	char*	(variable)

Table 2: SWE Common Data Model data types used within the SWE Bridge.

Note that Category is equivalent to text, but it only accepts a values from a pre-defined list. E.g., the interface category type can only be "TCP", "UDP" or "serial".

2 Principles of Operation

The SWE Bridge operation is organized in four components that are executed sequentially: the OGC PUCK Detector & Extractor, the decoder, the SensorML Interpreter and the Mission Scheduler. The execution of the SWE Bridge may vary slightly depending on whether the sensor is OGC PUCK-enabled or not, as shown in figure 1. OGC PUCK-enabled sensors shall provide their Sensor Deployment File (SDF, see section 2.1) embedded within their payload memory. However, when interfacing a commercial off-the-shelf sensor (COTS) its associated SDF shall be uploaded to the platform filesystem.

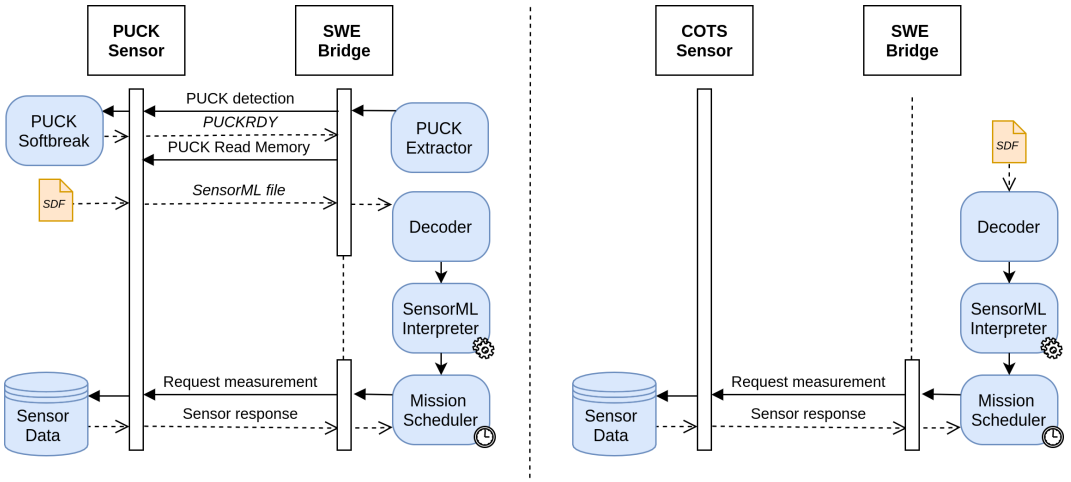


Figure 1: SWE Bridge execution diagram for PUCK-enabled and commercial off-the-shelf sensors (COTS)

The OGC PUCK Detector and Extractor detects new OGC PUCK-enabled sensors connected to the communications interface. Once a new sensor is detected, this component extracts its SDF. When interfacing a COTS sensor, a local SDF can be passed as argument to the SWE Bridge, bypassing the OGC PUCK Detector and Extractor component.

The decoder extracts the desired information from the SDF, which can be encoded in plain XML or in the efficient EXI format. The SensorML Interpreter service takes the extracted information and uses this data to configure the data acquisition workflow. If not configured by the PUCK extractor, the communication interface is also setup according to the information decoded sensor description. Afterwards, using the information from the sensor mission, the scheduler executes the data acquisition workflow, managing the sensor and storing sensor data.

2.1 Sensor Deployment File

The Sensor Deployment File (SDF) is a particular type of SensorML description file which includes all the information required to integrate a sensor on-the-fly using the SWE Bridge. This metadata file plays a key role, since it contains all the information to abstract the sensor characteristics and setup the acquisition process. The information required includes the sensor instance and the sensor mission.

The sensor instance is all the information that is intrinsic to a sensor and its deployment: its communication interface, the sensor protocol (set of commands), identifiers, deployment position, etc.

On the other hand, the sensor mission is the information that is required to setup an acquisition chain using the SWE Bridge. It includes which SWE Bridge modules should be used (see section 2.4) and how the information should flow between these modules. In other words, how they are connected.

2.2 Hardware Resources

The SWE Bridge aims to be integrated into any platform, regardless of its underlying hardware resources. In order to achieve a cross-platform implementation, the SWE Bridge has been implemented using ANSI C, with special emphasis on minimizing the usage of underlying software and hardware resources. All platform-dependent resources are abstracted using resource abstraction wrappers, which provide a unified way to access the platform's resources (see figure 2). These wrappers are the only functions that need to be adapted when deploying the SWE Bridge in a new platform.

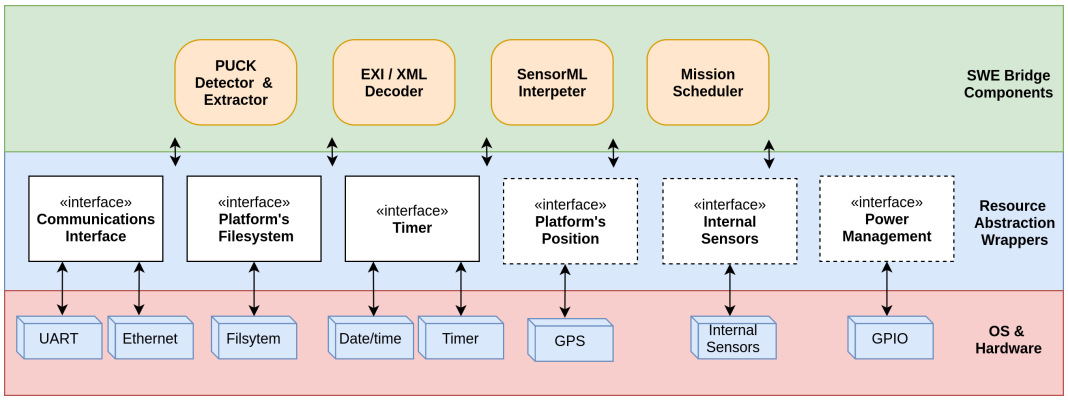


Figure 2: Sensor Deployment File overview

The boxes with dashed lines represent the optional parameters that can be implemented, depending on the observation platform. In example, an observation

platform that is deployed in a fix position does not require to implement the GPS wrappers.

2.3 SWE Bridge Modules

Modules are built-in implementations of generic operations within the SWE Bridge. Each module implements a generic operation, such as send a command, store data to a file, power on/off a sensor, etc. These modules can be instantiated within the SWE Bridge setup to generate processes, which can be connected generating a workflow. The term "process" is used when a module is called and loaded into the SWE Bridge's memory. As an analogy with object-oriented programming languages, "module" is a "class" while a "process" is an "object". The following modules are implemented within the SWE Bridge:

1. **Instrument Command**: Sends and/or receives a command from the sensor
2. **Insert Result**: Stores sensor data into a insert result file (O&M data)
3. **CSV Generator**: Generates CSV files from sensor data
4. **Linear Calibration**: Applies calibration parameters to sensor data
5. **Subsampling**: Subsamples incoming data
6. **Power Management**: Turns on/off sensors (requires GPIO hardware)
7. **Internal Sensors**: Reads platform's internal sensors (requires hardware internal sensors)
8. **Analog Measurement**: Performs an analog measurements (requires ADC hardware)
9. **Zabbix Sender**: Sends data to a Zabbix monitoring service (requires zabbix sender module)
10. **Hydrophone Stream**: receives high frequency streams from a hydrophone
11. **Sound Pressure Level**: Calculates SPL, RMS and SEL levels from hydrophone input
12. **WAV Generator**: Generates WAV files from hydrophone input
13. **WAV Reader**: Reads acoustic data from WAV files

2.4 Generating Acquisition Workflows

To generate a workflow the first step is to instantiate a set of modules. When a module is instantiated, a process is generated and loaded into the SWE Bridge Scheduler. These processes can be connected in a process chain, generating complex workflows.

The processes in a workflow pass data from source to destination (left to right in the diagrams in this section). Some processes generate data (e.g. instrument command), others require previous data to operate correctly (e.g. insert result) and some are even data independent (power management).

Although there are no restrictions in which processes can be connected, some configuration may not be valid. For instance, if it is not possible to connect a data-less process (e.g. power on/off sensor) to a process that requires input data (e.g. insert result). However, it is possible to connect processes that generate data to a process that does not require data (input data will be ignored).

In figure 3 a simple workflow is depicted. Two modules are instantiated, generating the "takeSample" process, and the "storeResult" process. The takeSample is an instrument command that periodically queries the sensor for data, and insert result stores this data into O&M files.

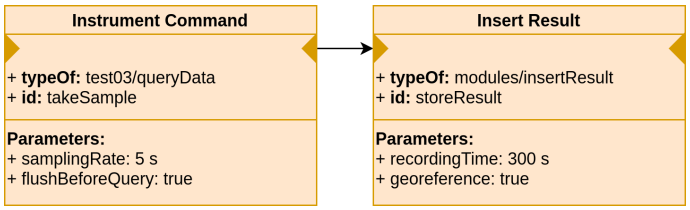


Figure 3: Example of a simple workflow

The previous workflow is very simple and only has two components, but using SWE Bridge module's much more complex workflows can be created. In figure 4 a complex workflow with 6 processes is depicted. This example models the scenario where the SWE Bridge periodically powers on the sensor, takes some measurements, and then powers it off again. Data gathered by the instrument command processed is stored in CSV format and in O&M format. Note that previously to the insert result process, the data is subsampled. Thus, data in O&M format will be a subsampled dataset, while CSV data will be the full dataset.

As a general rule, a process' output can be connected to several processes inputs. However, in order to maintain the data structure compatibility, a process can accept inputs from only one process as shown in figure 5.

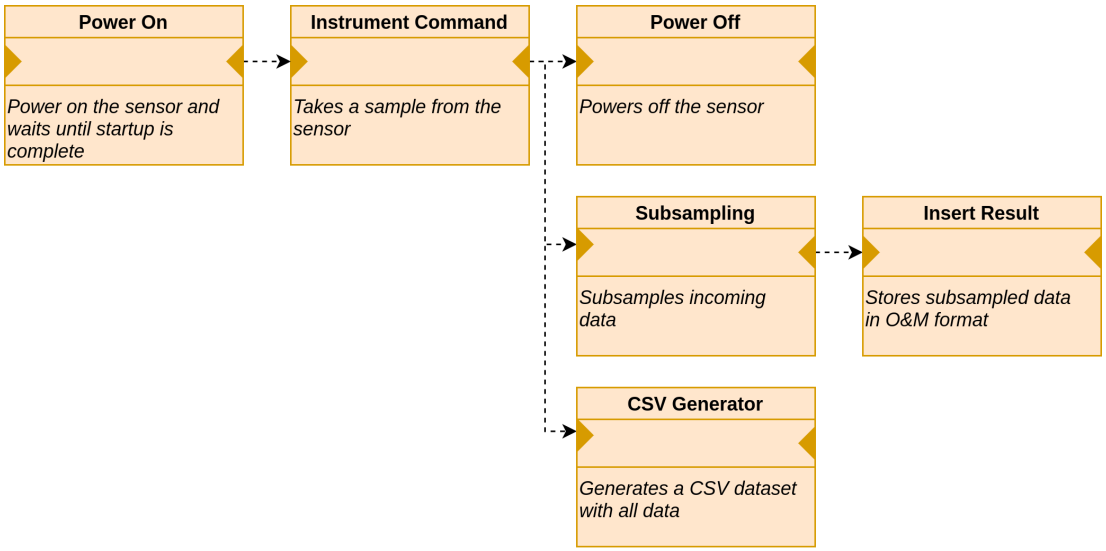


Figure 4: Example of a complex workflow

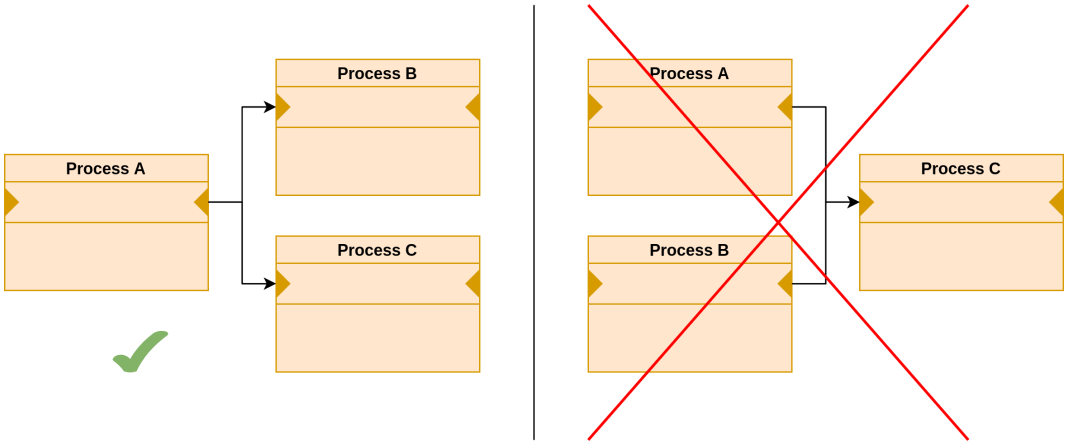


Figure 5: A process output can be connected to several inputs, but an input can only have one source.

2.5 Scheduler and Execution Conditions

Within the SWE Bridge all the processes (or module instances) are managed by the scheduler. The scheduler is the software component in charge of checking when and how a process should be executed. These execution conditions are managed through a set of flags named "execution modes". There are mainly four execution modes flags:

1. **initialization**: The process is executed once at startup

2. **interface interrupt**: Execution when incoming data is detected at the communication's interface

3. **scheduled process**: The processed is executed periodically

4. **previous process**: A previous process in the workflow triggers the execution of the next connected process

In addition to the four mentioned execution modes, there are other internally managed execution conditions. These extra conditions are used internally by the SWE Bridge to deal with timeouts and unfinished tasks in a non-blocking fashion.

Any process that has an associated interface (such as instrument command, hydrophone stream, etc.) will have its interface interrupt flag automatically set to true. No further configuration is required.

Scheduled process will be periodically executed by the scheduler. This period can be configured with the parameter `samplingRate`. Some modules also have aliases for this parameter to be more module-specific. In example, the `recordingTime` parameter in the insert result module. is in fact an alias for the `samplingRate`.

When generating a workflow, it is important to check when the execution will begin. If the mission is acquiring data from a sensor in streaming mode, the process chain may be started every time a packet is received at the communication's interface (interface interrupt). On the contrary, when the acquisition chain has to be executed periodically (e.g. query a sensor for data), the first process should be a scheduled process with an appropriate sampling rate.

3 Sensor Deployment File

3.1 Overview

The Sensor Deployment File (SDF) is a particular type of SensorML description file which includes all the information required to integrate a sensor on-the-fly using the SWE Bridge. The information required includes the sensor's interface, the sensor commands, a set of modules to operate the sensor and how these modules are connected to each other (see figure 6). Using the SWE Bridge Modules it is possible to generate data workflows, from very simple operations (query the sensor, store the result), to complex workflows including tens of different components.

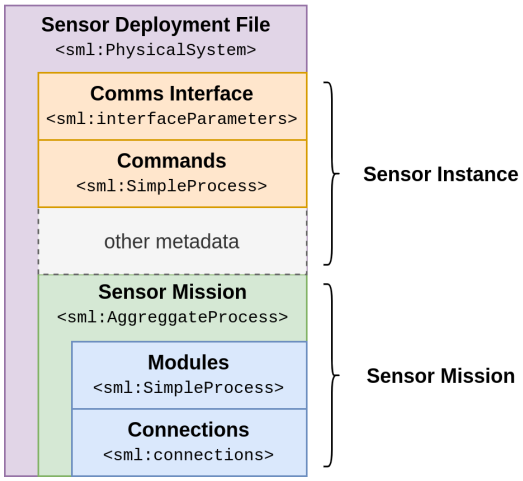


Figure 6: Sensor Deployment File overview

An SDF in separated in two different sections, the sensor instance and the sensor mission. The sensor instance includes the metadata that is intrinsic to the sensor, such as the commands (sensor protocol), its communications interface, deployment position, identifiers, contact information, etc. Although there is a lot of information that may be useful, only two elements within the sensor instance are required: the interface, and the sensor commands.

On the other hand, the sensor mission is related with how the data will be acquired and processed. It is not directly related to the sensor itself, but it tells the SWE Bridge how to process, manage and store the sensor's data. It has at least two sections, the modules and the connections. In the modules section, a set of functionalities within the SWE Bridge are invoked. Each module implements a specific function, e.g. take a measurement, store it in a file, power on/off a sensor, etc. The connections section tell the SWE Bridge how the data is passed from process to process, effectively arranging them into a workflow.

Figure 7 shows a compacted UML diagram of an SDF. Some intermediate SensorML elements are ommited to enhance readability, for more details see the code snippets. As it can be seen in the figure, the Sensor Mission consist of only two parts, the instantiation of modules and their connections. The sensor instance is composed mainly of two parts: the interface and a set of commands to define the sensor protocol.

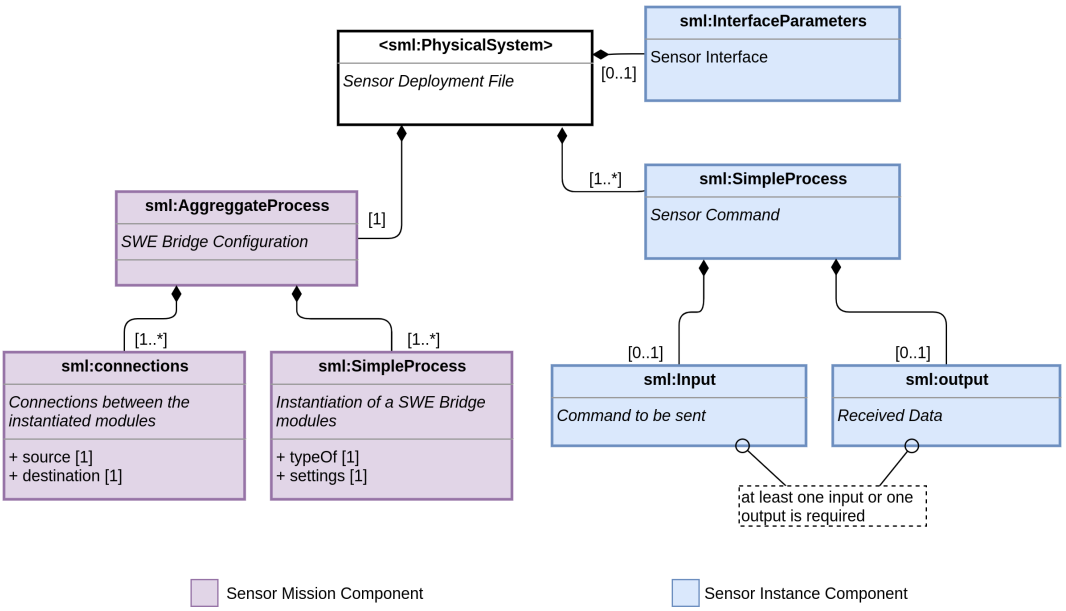


Figure 7: Sensor Deployment File UML diagram

3.2 Communication's Interface

In order to communicate with an instrument a communication's interface has to be configured. This configuration is performed according to the instrument's SensorML description as a `sml:parameter` section using a `sml:DataInterface` and `sml:interfaceParameters` element. Within the `sml:interfaceParameters` element a field with `portType` as name attribute defines which interface is being configured. The available protocols are Serial (or RS232), TCP/IP and UDP.

Note that if OGC PUCK protocol has been activated on a specific serial port, the interface configuration at the SensorML file will be ignored as the interface will be already configured.

3.2.1 Serial

Configures a Serial/RS232/UART connection, by default 8N1 (8 data bits, no parity and one stop bit).

Port Type: "RS232" or "UART"
Required parameters: "baudRate" (Count), "serialDevice" (Text)
Optional parameters: "softwareFlowControl" (Boolean, FALSE by default)

Example:

Listing 1: Example of RS232 / serial interface

```
<sml:parameter name="dataInterface">
  <sml:DataInterface id="serial_interface">
  <sml:data/>
  <sml:interfaceParameters>
    <swe:DataRecord>
    <!-- Interface Type -->
    <swe:field name="portType">
      <swe:Category>
      <swe:label>Port Type</swe:label>
      <swe:value>RS232</swe:value>
      </swe:Category>
    </swe:field>
    <swe:field name="serialDevice">
      <swe:Text>
      <swe:label>Serial Device</swe:label>
      <swe:value>/dev/ttyUSB3</swe:value>
      </swe:Text>
    </swe:field>
    <swe:field name="baudRate">
      <swe:Count>
      <swe:label>Baud Rate</swe:label>
      <swe:value>38400</swe:value>
      </swe:Count>
    </swe:field>
    <!-- Optional Fields -->
    <swe:field name="softwareFlowControl">
      <swe:Boolean>
      <swe:value>TRUE</swe:value>
    </swe:Boolean>
    </swe:field>
    </swe:DataRecord>
  </sml:interfaceParameters>
  </sml:DataInterface>
</sml:parameter>
```

3.2.2 UDP

Configures an UDP socket to receive streams. This interface does not allow to send commands since it will only open a socket to listen incoming packets.
Port Type: "UDP"
Required parameters:"portNumber" (Count)

Example:

Listing 2: Example of UDP interface

```
<sml:parameter name="dataInterface">
  <sml:DataInterface id="UDP_interface">
```

```
  <sml:data/>
  <sml:interfaceParameters>
    <swe:DataRecord>
    <swe:field name="portType">
      <swe:Category>
      <swe:label>Port Type</swe:label>
      <swe:value>UDP</swe:value>
      </swe:Category>
    </swe:field>
    <swe:field name="portNumber">
      <swe:Count>
      <swe:value>54321</swe:value>
      </swe:Count>
    </swe:field>
    </swe:DataRecord>
  </sml:interfaceParameters>
  </sml:DataInterface>
</sml:parameter>
```

3.2.3 TCP/IP

Configures a TCP/IP connection to send and receive packets.
Port Type: "TCP"
Required parameters: "IP" (Category), "portNumber" (Count)

Example:

Listing 3: Example of TCP/IP interface

```
<sml:parameter name="dataInterface">
  <sml:DataInterface id="TCP_interface">
  <sml:data/>
  <sml:interfaceParameters>
    <swe:DataRecord>
    <swe:field name="portType">
      <swe:Category>
      <swe:label>Port Type</swe:label>
      <swe:value>TCP</swe:value>
      </swe:Category>
    </swe:field>
    <swe:field name="portNumber">
      <swe:Count>
      <swe:value>54321</swe:value>
      </swe:Count>
    </swe:field>
     <swe:field name="IP">
     <swe:Category>
       <swe:value>127.0.0.1</swe:value>
     </swe:Category>
     </swe:field>
   </swe:DataRecord>
  </sml:interfaceParameters>
  </sml:DataInterface>
</sml:parameter>
```

3.3 Sensor Commands

The sensor's communication protocol is defined through a set of commands, each one encoded within a `sml:SimpleProcess` element. Each command is composed by one input and/or one output, depending on its nature. An input represents data sent from the SWE Bridge to the sensor, while the output represent data from the sensor to the SWE Bridge. There are three types of commands, depending on if they have an input, an output or both:

- Input and Output: A command is sent to the sensor, and a reply is expected (query).
- Only input: A command is sent to the sensor, but the sensor does not reply.
- Only output: the sensor periodically sends data without any input (data stream)

It is important that all the commands inherit the properties of the SWE Bridge instrument command module by adding the following line:

```
<sml:typeOf xlink:title="swebridge:modules:instrumentCommand"/>
```

Additionally, the `sml:SimpleProcess` should have a `gml:identifier` providing a unique id for the command. This identifier is important, since it will be used to reference the command from the sensor mission:

```
<gml:identifier codeSpace="uid">test01:dataStream</gml:identifier>
```

3.3.1 Data Model

As shown in figure 7, each sensor command is composed of an input and/or and output. Each input models the data to be sent to the sensor (e.g. configuration command, query command). Each output models the data sent by the sensor to the SWE Bridge. Each input/output has mainly two parts, the data model and the encoding. The data model (encoded within a `swe:elementType` element) represents the meaning of each data segment, while the encoding explicits the format of this data (binary, ASCII, etc.). Inputs and outputs, shown in figure 8 are almost identical, the main difference is that the input requires a `swe:value` element.

3.3.1.1 Fields

The data model is composed by a collection of `swe:field` elements. Each field describes a part of the data stream and it is composed by the following elements:

- **name**: human-readable name that will be assigned to the variable

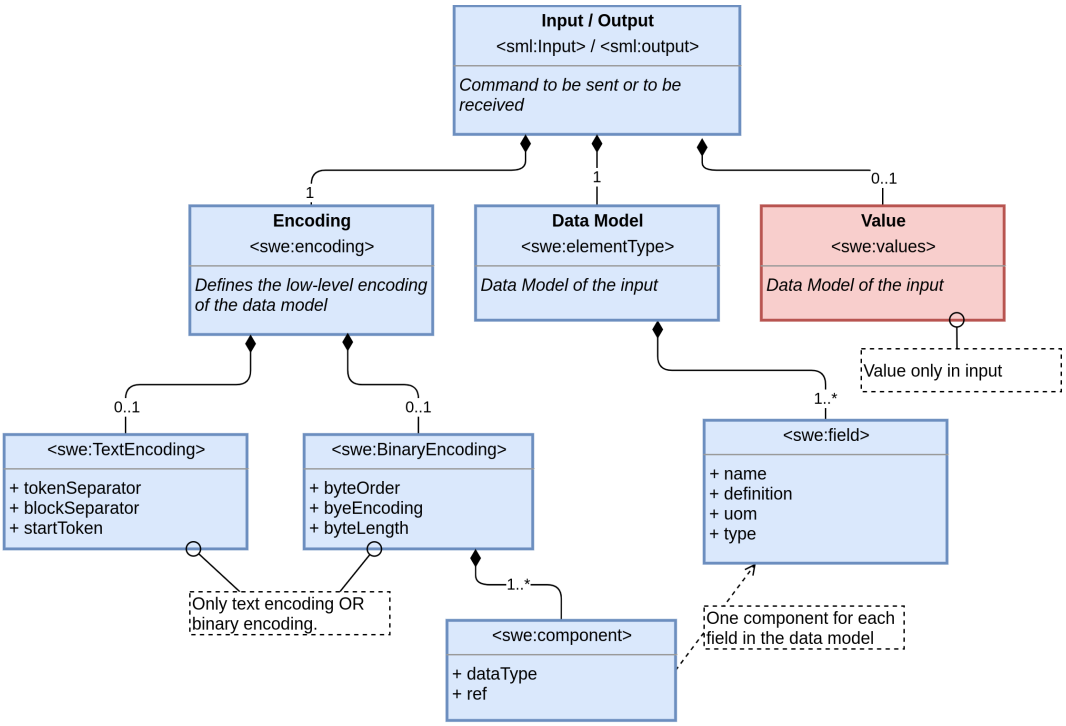


Figure 8: Input / Output UML model

- **definition** (optional): Link to an online resource defining the variable. It is encouraged to use controlled vocabularies
- **type**: The Data type of the variable is the field's children (see section 1.2)
- **uom**: Units of Measurement of the variable. Only if the type is Quantity (floating-point value).

3.3.1.2 Data Record

In order to group several `swe:field` elements, a `swe:DataRecord` element is required. Consider an ASCII data string coming from a CTD sensor which includes temperature, conductivity and depth as floating-point values separated by a comma:

```
RX: <temperature>,<conductivity>,<depth><CR><NL>
```

In example:

```
RX: 13.5475,4.7894,23.8964<CR><NL>
```

Each variable (temperature, conductivity, depth) has to be encoded in its own `swe:field` element. The field's children is the element type, in this case a floating-point value (Quantity). When using multiple fields, they have to be grouped using a `swe:DataRecord` element:

Listing 4: Data Model using a data record

```
<swe:DataRecord>
  <swe:field name="sea_water_temperature">
    <swe:Quantity
      definition="http://mmisw.org/ont/cf/parameter/sea_water_temperature">
      <swe:uom code="degC"/>
    </swe:Quantity>
  </swe:field>
  <swe:field name="conductivity">
    <swe:Quantity
      definition="http://mmisw.org/ont/cf/parameter/sea_water_electrical_conductivity">
      <swe:uom code="S/m"/>
    </swe:Quantity>
  </swe:field>
  <swe:field name="depth">
    <swe:Quantity
      definition="http://mmisw.org/ont/cf/parameter/depth">
      <swe:uom code="m"/>
    </swe:Quantity>
  </swe:field>
</swe:DataRecord>
```

3.3.1.3 Data Array

Some instruments send similar parameters tens or hundreds of times (e.g. ADCPs), so using a field for each one of them is not practical. In these cases a `swe:DataArray` can be used. The DataArray consists of a number of repeated elements, all of the same type. It has two components: `swe:elementCount` and `swe:elementType`. The element count define how many times the contents of the array are repeated. The element type defines the contents of each instance of the array by using a `DataRecord` to encode the individaual components of each array member:

Listing 5: Data Model using a data array

```
<swe:DataArray id="ArrayID">
  <swe:elementCount xlink:href="#arrayCount"/>
  <swe:elementType name="outputArray" xlink:title="ElementTypeTitle">
    <swe:DataRecord>
      <swe:field name="field1">
        <swe:Quantity definition="http://.../param1">
          <swe:uom code="m/s"/>
        </swe:Quantity>
      </swe:field>
      <swe:field name="field2">
        <swe:Quantity definition="http://.../param2">
          <swe:uom code="m/s"/>
        </swe:Quantity>
      </swe:field>
      ...
```

```
      </swe:DataRecord>
    </swe:elementType>
  </swe:DataArray>
```

Note that the number of elements in the array is not provided in the example, but referenced using the `xlink:href` attribute. It is common practice to define the element count as a parameter within the `sml:SimpleProcess`:

Listing 6: Definition of the array count as a parameter

```
<sml:parameter name="arrayCountParameter" >
  <swe:Count id="arrayCount">
    <swe:value>3</swe:value>
  </swe:Count>
</sml:parameter>
```

Data arrays can be nested within data records to create more complex structures that include non-repeating and repeating parts.

3.3.2 Encoding

The previous section explaind how to encode a data model, but the low-level encoding details are not provided. The encoding section provides such details. As depicted in figure 8, there are mainly two types of encoding binary and text.

3.3.2.1 Text Encoding Text encoding is used for sensors using plain ASCII communications. The parameters in a `swe:TextEncoding` are:

- **tokenSeparator**: token that separates each component in the data stream
- **blockSeparator**: token that indicates end of the data stream.
- **startToken** (optional): token that indicates the start of a data stream
- **collapseWhiteSpaces** (optional): indicates if white spaces should be ignored during the command parsing (space, tab, carriage return, new line).

In the CTD example mentioned before, the three values are separated by a comma and the transmission block ends with a carriage return and line feed. In this case, the token separator is a comma, the block separator is carriage return and new line:

Listing 7: Text Encoding example

```
<swe:encoding>
  <swe:TextEncoding tokenSeparator="," blockSeparator="&#x000D;&#x0000A;"/>
</swe:encoding>
```

Although not defined in the SWE Common data Model standard, the SWE Bridge accepts the definition of a start token. Start token is a string that indicates the start of a valid data stream. Let's consider the previous example stream, but with a leading $ character:

```
RX: $<temperature>,<conductivity>,<depth><CR><NL>
```

To encode this stream, it is possible to use the `swe:extension` to place a user-defined startToken text element. It is important to maintain the id attribute as "startToken", so the SWE Bridge can recognize it:

Listing 8: Text Encoding with startToken

```
<swe:encoding>
  <swe:TextEncoding tokenSeparator="," blockSeparator="&#x000D;&#x0000A;"
                    collapseWhiteSpaces="True">
    <swe:extension>
      <swe:Text id="startToken">
        <swe:value>$</swe:value>
      </swe:Text>
    </swe:extension>
  </swe:TextEncoding>
</swe:encoding>
```

Note that the start token element is expected at the beginning of the data stream. Also note that token separator is not expected within the start token and the first element. If a start token is specified, all incoming characters received before the start token will be automatically discarded.

If block separator is empty, the SWE Bridge will store all incoming characters in a buffer and will attempt to process them once the timeout timer expires.

3.3.2.2 Binary Encoding Binary communications are encoded into data packets with a fixed size where each component of the data stream has a data tyep(32-bit integer, float64, unsigned byte, etc) and can be mapped to an element within the data model. When using binary communications the following attributes of the binary stream have to be provided:

- **byteOrder**: littleEndian or bigEndian
- **byteEncoding**: raw or base64 (only used to encode binary streams into plain text)
- **byteLength**: total number of bytes in the data stream

After providing the general details, the encoding of each element in the data model has to be provided using a `swe:component` element. Each component has the following attributes:

- **dataType**: type of the data (integer, float, byte, etc.)

- **ref**: reference to the data model element being described

Let's imagine that we have the following binary data stream, composed by the three elements:

```
RX: <a (uint32)><b (int64)><c (float32)>
```

The encoding of each component of the data model (represented in a `swe:field`) has to be specified using a component, filling the `dataType` and a reference (`ref`) to the field in the data model.

Listing 9: Output modelling a binary command

```
<swe:DataStream>
  <swe:elementType name="dataModel">
    <swe:DataRecord>
      <!-- Define parameter a -->
      <swe:field name="a">
        <swe:Count definition="http://some.vocab.com/a"/>
      </swe:field>
      ...
    </swe:DataRecord>
  </swe:elementType>
  <swe:encoding>
    <!-- Set general parameters of the data stream -->
    <swe:BinaryEncoding byteOrder="littleEndian" byteEncoding="raw" byteLength="16">
      <swe:member>
        <!-- Define a's dataType and reference it to the data model -->
        <swe:Component
          dataType="http://www.opengis.net/def/dataType/OGC/0/unsignedInt"
          ref="dataModel/a"/>
      </swe:member>
      ...
    </swe:BinaryEncoding>
  </swe:encoding>
  <!-- values are not encoded as they depend on the sensor response -->
  <swe:values/>
</swe:DataStream>
</sml:output>
```

Note that in the reference to each component of the data model is composed by the `swe:elementType`'s and `swe:field`'s name attributes joined with a slash.

The complete list of the data types is available at the SWE Data Common Standard, section 8.6.1 [3].

3.3.3 Input Commands

When encoding an input (command to be sent to the sensor) the easiest way is to define the whole command as a single `swe:Text` field and set its value as a string. Then, the value of the command can be easily set using the `swe:values` element. In the case of text encoding the block separator will be added at the end of the transmission.

Listing 10: Input using text encoding

```
<sml:input name="command">
<sml:DataInterface>
  <sml:data>
  <swe:DataStream>
    <!-- The element type defines the data model -->
    <swe:elementType name="dataModel">
    <swe:DataRecord>
      <swe:field name="text">
      <swe:Text/>
      </swe:field>
      <!-- NOTE: A block separator is always added at the end of the DataRecord -->
    </swe:DataRecord>
    </swe:elementType>
    <!-- Encoding of the command data model -->
    <swe:encoding>
    <swe:TextEncoding tokenSeparator="" blockSeparator="&#x0D;&#x0A;"/>
    </swe:encoding>
    <!-- Set the values to the data model (separated by tokenSeparator)-->
    <swe:values>TakeSample?</swe:values>
  </swe:DataStream>
  </sml:data>
</sml:DataInterface>
</sml:input>
```

In binay communications a similar approach can be used: a data array with a variable length is defined. Then, in the `swe:values` element, the binary command is embedded. However, in order to encode binary data into SensorML file the base64 has to be used. Base64 is a special codification that permits the encoding of hexadecimal data chunks into plain-text formats such as XML. There are multiple online tools to convert from binary to base64 and vice-versa.

Listing 11: Input using binary encoding

```
<sml:input name="command">
  <sml:DataInterface>
    <sml:data xlink:type="simple">
      <swe:DataStream>
        <swe:elementType name="command" xlink:type="simple">
          <swe:DataArray>
            <!-- element count not specified, it's the length of the list -->
            <swe:elementCount/>
            <swe:elementType name="bytes">
              <swe:Count/>
            </swe:elementType>
          </swe:DataArray>
        </swe:elementType>
        <swe:encoding>
          <swe:BinaryEncoding byteOrder="littleEndian" byteEncoding="base64">
            <swe:member>
              <swe:Component dataType="http://www.opengis.net/def/dataType/OGC/0/unsignedByte"
                             ref="command/bytes"/>
            </swe:member>
          </swe:BinaryEncoding>
        </swe:encoding>
        <!-- Hex values: 0x00 0x01 0x02 0x03 -->
        <swe:values>AAECAw==</swe:values>
      </swe:DataStream>
```

```
    </sml:data>
  </sml:DataInterface>
</sml:input>
```

3.3.4 Sensor Commands Examples

This example encodes a Sensor Command from a CTD sensor streaming temperature, conductivity and

```
<temperature>,<conductivity>,<depth><CR><NL>
```

In example:

```
13.5475,4.7894,23.8964<CR><NL>
```

Listing 12: Data stream example in ASCII encoding

```
<sml:SimpleProcess gml:id="queryDataID">
  <sml:typeOf xlink:title="swebridge:modules:instrumentCommand"/>
  <gml:identifier codeSpace="uid">test01:dataStream</gml:identifier>
  <sml:outputs>
    <sml:OutputList>
      <sml:output name="dataOut">
        <sml:DataInterface>
          <sml:data>
            <swe:DataStream>
            <!-- Define the data model -->
              <swe:elementType name="response">
                <swe:DataRecord>
                  <swe:field name="sea_water_temperature">
                    <swe:Quantity
                      definition="http://.../sea_water_temperature">
                      <swe:uom code="degC"/>
                    </swe:Quantity>
                  </swe:field>
                  <swe:field name="conductivity">
                    <swe:Quantity
                      definition="http://.../sea_water_electrical_conductivity">
                      <swe:uom code="S/m"/>
                    </swe:Quantity>
                  </swe:field>
                  <swe:field name="depth">
                    <swe:Quantity
                      definition="http://mmisw.org/ont/cf/parameter/depth">
                      <swe:uom code="m"/>
                    </swe:Quantity>
                  </swe:field>
                </swe:DataRecord>
              </swe:elementType>
              <!-- Define the response text encoding -->
              <swe:encoding>
                <swe:TextEncoding tokenSeparator="," blockSeparator="&#x000D;&#x0000A;"/>
              </swe:encoding>
              <swe:values/>
            </swe:DataStream>
```

```
        </sml:data>
      </sml:DataInterface>
    </sml:output>
  </sml:OutputList>
 </sml:outputs>
</sml:SimpleProcess>
```

The following example models a query command, the SWE Bridge sends a command and a response is expected:

```
TX: takeSample?<CR><NL>
RX: <temperature>,<conductivity>,<depth><CR><NL>
```

Listing 13: Example of an plain text query command

```
<sml:SimpleProcess gml:id="queryData">
  <gml:identifier codeSpace="uid">test03:queryData</gml:identifier>
  <sml:typeOf xlink:title="swebridge:modules:instrumentCommand"/>
  <sml:inputs>
    <sml:InputList>
      <!-- The input corresponds to the command that will be sent to the sensor -->
      <sml:input name="command">
        <sml:DataInterface>
          <sml:data>
            <swe:DataStream>
              <!-- The element type defines the data model -->
              <swe:elementType name="dataModel">
                <swe:DataRecord>
                  <swe:field name="text">
                    <swe:Text/>
                  </swe:field>
                  <!-- NOTE: A block separator is always added at the end of the DataRecord -->
                </swe:DataRecord>
              </swe:elementType>
              <!-- Encoding of the command data model -->
              <swe:encoding>
                <swe:TextEncoding tokenSeparator="" blockSeparator="&#x0D;&#x0A;"/>
              </swe:encoding>
              <!-- Set the values to the data model (separated by tokenSeparator)-->
              <swe:values>TakeSample?</swe:values>
            </swe:DataStream>
          </sml:data>
        </sml:DataInterface>
      </sml:input>
    </sml:InputList>
  </sml:inputs>
  <sml:outputs>
    <sml:OutputList>
      <!--The output corresponds with the sensor's response -->
      <sml:output name="dataOut">
        <sml:DataInterface>
          <sml:data>
            <swe:DataStream>
              <!-- the elementType is a container for the data model -->
              <swe:elementType name="reply">
                <!-- Defining the output (both header and array) as a DataRecord -->
                <!-- NOTE: The DataRecords are separated by blockSeparator -->
                <swe:DataRecord>
```

```
            <!-- Header, modelled as a DataRecord containing 3 fields -->
            <!-- Fields within the header "DataRecord" are separator by a token separator -->
            <swe:field name="header">
              <swe:DataRecord definition="def" id="DataRecordID">
                <!-- Header field 1 -->
                <swe:field name="field1">
                  <swe:Quantity>
                    <swe:uom code="glurps"/>
                  </swe:Quantity>
                </swe:field>
                <swe:field name="field2">
                  <swe:Quantity>
                    <swe:uom code="glurps"/>
                  </swe:Quantity>
                </swe:field>
              </swe:DataRecord>
            </swe:field>
          </swe:DataRecord>
        </swe:elementType>
        <!-- Define the Encoding of the array -->
        <!-- token separator ',' (between fields within a member) -->
        <!-- block separator '\r\n' (between different elements) -->
        <!-- collapseWhiteSpaces indicate if all "white" spaces (CR, LF, TAB, space)
          surrounding the token and block separators should be ignored -->
        <swe:encoding>
          <swe:TextEncoding tokenSeparator="," blockSeparator="&#x0D;&#x0A;"
                            collapseWhiteSpaces="true"/>
        </swe:encoding>
        <swe:values/>
      </swe:DataStream>
    </sml:data>
  </sml:DataInterface>
 </sml:output>
</sml:OutputList>
</sml:outputs>
</sml:SimpleProcess>
```

The following example models a binary query command, the SWE Bridge sends a command, and a response is expected. The sensor responds has a leading field (named "count"), followed by an array of three elements (uint, float64 and signed byte) repeated 10 times.

```
TX:0x01 0x02 0x03 0x04
RX: <count>{<uint><float64><byte>}x10
```

Listing 14: Example of binary query with an nested array

```
<sml:SimpleProcess gml:id="queryData">
  <gml:identifier codeSpace="uid">test05:queryData</gml:identifier>
  <sml:typeOf xlink:title="swebridge:modules:instrumentCommand"/>
  <sml:inputs>
    <sml:InputList>
      <!-- Command that will be sent to sensor -->
      <sml:input name="command">
        <sml:DataInterface>
          <sml:data xlink:type="simple">
            <swe:DataStream>
```

```
            <swe:elementType name="command" xlink:type="simple">
              <swe:DataArray>
                <!-- element count not specified, it's the length of the list -->
                <swe:elementCount/>
                <swe:elementType name="bytes">
                  <swe:Count/>
                </swe:elementType>
              </swe:DataArray>
            </swe:elementType>
            <swe:encoding>
              <swe:BinaryEncoding byteOrder="littleEndian" byteEncoding="base64">
                <swe:member>
                  <swe:Component ref="command/bytes"
                       dataType="http://www.opengis.net/def/dataType/OGC/0/unsignedByte"/>
                </swe:member>
              </swe:BinaryEncoding>
            </swe:encoding>
            <!-- Hex values: 0x00 0x01 0x02 0x03 -->
            <swe:values>AAECAw==</swe:values>
          </swe:DataStream>
        </sml:data>
      </sml:DataInterface>
    </sml:input>
  </sml:InputList>
</sml:inputs>
  <sml:outputs>
  <sml:OutputList>
    <!-- Sensor Response -->
    <sml:output name="test1Output">
      <sml:DataInterface>
        <sml:data>
          <swe:DataStream>
            <swe:elementType name="dataModel">
              <swe:DataRecord>
                <!-- Leading count element -->
                <swe:field name="count">
                  <swe:Count/>
                </swe:field>
                <!-- nested array -->
                <swe:field name="array">
                  <swe:DataArray>
                    <swe:elementCount xlink:href="#arrayCount"/>
                    <swe:elementType name="model" xlink:type="simple">
                      <swe:DataRecord>
                        <swe:field name="unsignedInt">
                          <swe:Count/>
                        </swe:field>
                        <swe:field name="float64">
                          <swe:Quantity>
                            <swe:uom code="Glurps"/>
                          </swe:Quantity>
                        </swe:field>
                        <swe:field name="signedByte">
                          <swe:Count/>
                        </swe:field>
                      </swe:DataRecord>
                    </swe:elementType>
                  </swe:DataArray>
                </swe:field>
```

```
            </swe:DataRecord>
          </swe:elementType>
          <swe:encoding>
            <swe:BinaryEncoding byteOrder="littleEndian" byteEncoding="raw" byteLength="96">
              <!-- As the data structure has a fixed size, the byteLength variable is added -->
                <swe:Component
                  dataType="http://www.opengis.net/def/dataType/OGC/0/unsignedInt"
                  ref="dataModel/count"/>
              </swe:member>
              <swe:member>
                <swe:Component
                  dataType="http://www.opengis.net/def/dataType/OGC/0/unsignedInt"
                  ref="dataModel/array/model/unsignedInt"/>
              </swe:member>
              <swe:member>
                <swe:Component
                  dataType="http://www.opengis.net/def/dataType/OGC/0/double"
                  ref="dataModel/array/model/float64"/>
              </swe:member>
              <swe:member>
                <swe:Component
                  dataType="http://www.opengis.net/def/dataType/OGC/0/signedByte"
                  ref="dataModel/array/model/signedByte"/>
              </swe:member>
            </swe:BinaryEncoding>
          </swe:encoding>
          <!-- values are not encoded as they depend on the sensor response -->
          <swe:values/>
        </swe:DataStream>
      </sml:data>
    </sml:DataInterface>
  </sml:output>
 </sml:OutputList>
 </sml:outputs>
<sml:parameters>
 <sml:ParameterList>
  <!-- Define the length of the nested DataArray as a parameter -->
  <sml:parameter name="p">
   <swe:Count id="arrayCount">
    <swe:value>10</swe:value>
   </swe:Count>
  </sml:parameter>
 </sml:ParameterList>
</sml:parameters>
</sml:SimpleProcess>
```

3.4 Sensor Mission

As described in section 2.1, it is not enough to define the sensor itself. In order to operate the SWE Bridge, its modules have to be instantiated, configured and connected. The sensor mission is defined as a `sml:AggregateProcess` within the sensor deployment file. The aggregate process has only two children: an array of processes (`sml:SimpleProcess`) and the connections between these processes (`sml:connections`).

3.4.1 Instantiating Modules

In order to instantiate a module, a `sml:component` with a `sml:SimpleProcess` is required. Within the simple process element, the `sml:typeOf` element is required to specify the type of process. The reference can point to a SWE Bridge module, or it can point to a previously defined sensor command (see section 3.3).

In this section the simple workflow depicted in figure 3 will be encoded in SensorML. The takeSample process can be instantiated using the following example, referencing the `test03:queryData` element.

Listing 15: Declaration of the takeSample process

```
<sml:component name="takeSample">
    <sml:SimpleProcess gml:id="storeResultId" >
        <!-- Inherit from previously declared command -->
        <sml:typeOf xlink:title="test03:queryData"/>
        <sml:configuration>
            <sml:Settings>
                <sml:setValue ref="parameters/samplingRate">5</sml:setValue>
                <sml:setValue ref="parameters/flushBeforeQuery">true</sml:setValue>
            </sml:Settings>
        </sml:configuration>
    </sml:SimpleProcess>
</sml:component>
```

Note that the `sml:typeOf` does not reference directly a SWE Bridge module, but refers to a sensor command previously defined. The `xlink:title` attribute points to the `gml:identifier` of a sensor command defined in the sensor instance (listing 13).

In order to fine-tune the process, the `sml:Settings` is used to set the appropriate module's configuration parameters. A complete list of each module's parameters is provided later in this section 5.

The following snippet shows to instantiate an insert result module:

Listing 16: Declaration of the storeResult process

```
<sml:component name="storeResult">
    <sml:SimpleProcess gml:id="storeResultId">
        <sml:typeOf xlink:title="swebridge:modules:insertResult"/>
        <sml:configuration>
            <sml:Settings>
                <sml:setValue ref="parameters/recordingTime">60.0</sml:setValue>
                <sml:setValue ref="parameters/template">swe_bridge_test03</sml:setValue>
                <sml:setValue ref="parameters/georeference">true</sml:setValue>
            </sml:Settings>
        </sml:configuration>
    </sml:SimpleProcess>
</sml:component>
```

In this example the `sml:typeOf` directly references the module. In this case the settings define the period for generating data files (recording time), the template identifier for the O&M file (template), and the option to georeference acquired data is set.

3.4.1.1 Execution Modes

In order to control the execution modes of each process, the execution modes have to be selected. Previous process and interface interrupt execution modes are automatically managed by the SWE Bridge. The only modes that need to be specified are `initialization` and `scheduled` process. Any process that has its input connected to another process' output will be automatically executed when the previous process is successfully executed. Similarly, when a packet

A process can be set to `initialization` by adding the setting its initialization flag within the process' `sml:Settings`:

```
<sml:setValue ref="parameters/executionModes/initialization">true</sml:setValue>
```

Any process can be converted to a `scheduled process` by setting its sampling rate to a positive value in seconds:

```
<sml:setValue ref="parameters/samplingRate">10.0</sml:setValue>
```

3.4.2 Connecting Modules

In order to connect the processes declared in listigns 15 and 16, the `sml:connection` element is required:

Listing 17: Connections from takeSample's output storeResult's input

```
<sml:connection>
    <sml:Link>
        <sml:source ref="components/takeSample/outputs/dataOut"/>
        <sml:destination ref="components/storeResult/inputs/dataIn"/>
    </sml:Link>
</sml:connection>
```

These connections follow the referencing rules for the SensorML standards. To simplify the referencing all SWE Bridge processes have only one input and one output, named "dataIn" and "dataOut" respectively. Thus, to reference a processes' input the following path has to be used:

`"component/<component name>/inputs/dataIn"`

The outputs are always referenced as:

`"component/<component name>/outputs/dataOut"`

3.5 Sensor Mission Examples

Within the SWE Bridge repository, there is a comprehensive set of 14 example covering most of the modules and options. Here, some of their test missions are discussed.

The SWE Bridge test mission 4 (figure 9) provides an example on how to generate two datasets from a sensor streaming data. One datset conains the full data, while the other is subsmpled (only one out of three samples is stored).

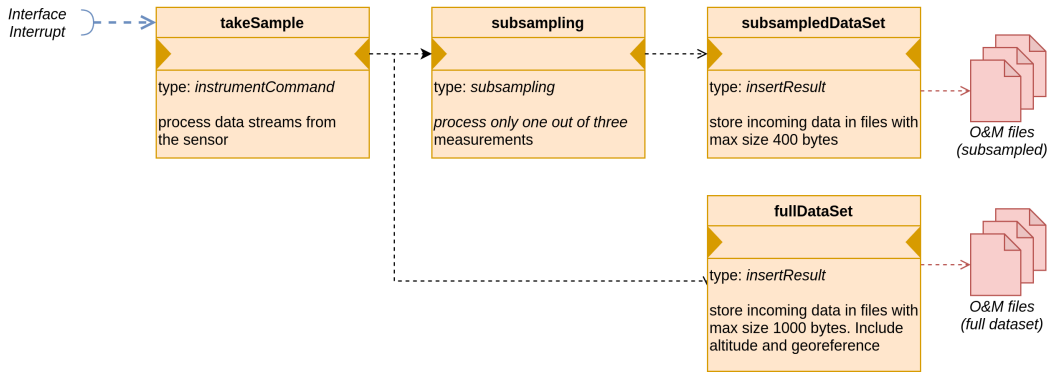


Figure 9: SWE Bridge test mission 4. From a sensor streaming data, two datasets are generated, one containing the full data, and another storing only one out of each three measurements

Listing 18: SWE Bridge test 4 mission

```
<sml:AggregateProcess gml:id="test04Mission">
  <sml:components>
    <sml:ComponentList>
      <sml:component name="takeSample">
        <sml:SimpleProcess gml:id="takeSample">
          <sml:typeOf xlink:title="test04:queryData"/>
        </sml:SimpleProcess>
      </sml:component>
      <sml:component name="subsampling">
        <sml:SimpleProcess gml:id="subsampling">
          <sml:typeOf xlink:title="swebridge:modules:subsampling"/>
          <sml:configuration>
            <sml:Settings>
              <sml:setValue ref="parameters/subsamplingRatio">3</sml:setValue>
            </sml:Settings>
          </sml:configuration>
        </sml:SimpleProcess>
      </sml:component>
      <sml:component name="fullDataSet">
        <sml:SimpleProcess gml:id="fullDataSet">
          <sml:typeOf xlink:title="swebridge:modules:insertResult"/>
          <sml:configuration>
            <sml:Settings>
              <sml:setValue ref="parameters/maxFileSize">1000</sml:setValue>
              <sml:setValue ref="parameters/template">swe_bridge_test04</sml:setValue>
              <sml:setValue ref="parameters/decimalPrecision">6</sml:setValue>
              <sml:setValue ref="parameters/georeference">true</sml:setValue>
              <sml:setValue ref="parameters/altitude">true</sml:setValue>
            </sml:Settings>
          </sml:configuration>
        </sml:SimpleProcess>
      </sml:component>
      <sml:component name="subsampledDataSet">
        <sml:SimpleProcess gml:id="subsampledDataSet">
          <sml:typeOf xlink:title="swebridge:modules:insertResult"/>
          <sml:configuration>
            <sml:Settings>
```

```
            <sml:setValue ref="parameters/maxFileSize">400</sml:setValue>
            <sml:setValue ref="parameters/EXIencoding">true</sml:setValue>
            <sml:setValue ref="parameters/template">swe_bridge_test04_subsampled</sml:setValue>
            <sml:setValue ref="parameters/decimalPrecision">6</sml:setValue>
            <sml:setValue ref="parameters/georeference">true</sml:setValue>
            <sml:setValue ref="parameters/altitude">true</sml:setValue>
          </sml:Settings>
        </sml:configuration>
      </sml:SimpleProcess>
    </sml:component>
  </sml:ComponentList>
 </sml:components>
 <sml:connections>
  <sml:ConnectionList>
    <!-- Connects the instrument output witht the selector component input-->
    <sml:connection>
      <sml:Link>
        <sml:source ref="components/takeSample/outputs/dataOut"/>
        <sml:destination ref="components/subsampling/inputs/dataIn"/>
      </sml:Link>
    </sml:connection>
    <sml:connection>
      <sml:Link>
        <sml:source ref="components/subsampling/outputs/dataOut"/>
        <sml:destination ref="components/subsampledDataSet/inputs/dataIn"/>
      </sml:Link>
    </sml:connection>
    <sml:connection>
      <sml:Link>
        <sml:source ref="components/takeSample/outputs/dataOut"/>
        <sml:destination ref="components/fullDataSet/inputs/dataIn"/>
      </sml:Link>
    </sml:connection>
  </sml:ConnectionList>
 </sml:connections>
</sml:AggregateProcess>
```

The SWE Bridge test mission 5 (figure 10) provides an example on how to manage the power cycle of a sensor. Every ten seconds the sensor is powered on and it is queried for data. Once the measurement has been acquired, the sensor is powered off and the data is stored in O&M file. The cycle is controlled by the first process' sampling rate (10 seconds).

Listing 19: SWE Bridge test 4 mission

```
<sml:AggregateProcess gml:id="test05Mission">
 <sml:components>
  <sml:ComponentList>
    <sml:component name="PowerOnSensor">
      <sml:SimpleProcess gml:id="PowerOnSensor" >
        <!-- Inherit from Sensor SimpleProcess where the outputs are defined -->
        <sml:typeOf xlink:title="swebridge:process:powerManagement"/>
        <sml:configuration>
          <sml:Settings>
            <!-- Connect the ADC to the fields -->
            <sml:setValue ref="parameters/delayAfter">3.0</sml:setValue>
            <sml:setValue ref="parameters/powerIndex">12</sml:setValue>
            <sml:setValue ref="parameters/power">ON</sml:setValue>
```

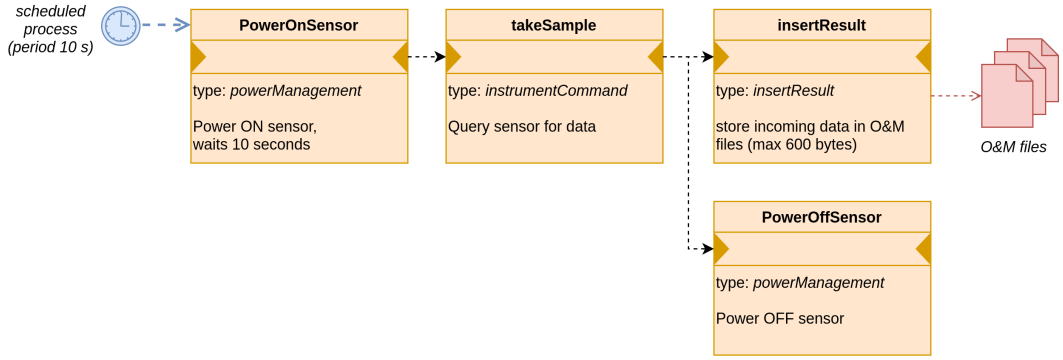


Figure 10: SWE Bridge test mission 5, showing how to manage a sensor's power cycle

```
        <sml:setValue ref="parameters/samplingRate">10.00</sml:setValue>
      </sml:Settings>
    </sml:configuration>
  </sml:SimpleProcess>
</sml:component>
<sml:component name="takeSample">
  <sml:SimpleProcess gml:id="takeSample" >
    <sml:typeOf xlink:title="test05:queryData"/>
    <sml:configuration>
      <sml:Settings>
        <sml:setValue ref="parameters/executionModes/previousProcess">TRUE</sml:setValue>
        <sml:setValue ref="parameters/watchdog">30.0</sml:setValue>
      </sml:Settings>
    </sml:configuration>
  </sml:SimpleProcess>
</sml:component>
<sml:component name="insertResult">
  <sml:SimpleProcess gml:id="insertResult">
    <gml:identifier codeSpace="uid">swebridge:modules:insertResult:1</gml:identifier>
    <sml:typeOf xlink:title="swebridge:modules:insertResult"/>
    <sml:configuration>
      <sml:Settings>
        <sml:setValue ref="parameters/maxFileSize">600</sml:setValue>
        <sml:setValue ref="parameters/template">swe_bridge_test05</sml:setValue>
        <sml:setValue ref="parameters/decimalPrecision">6</sml:setValue>
        <sml:setValue ref="parameters/georeference">true</sml:setValue>
        <sml:setValue ref="parameters/altitude">true</sml:setValue>
      </sml:Settings>
    </sml:configuration>
  </sml:SimpleProcess>
</sml:component>
<sml:component name="PowerOffSensor">
  <sml:SimpleProcess gml:id="PowerOffSensor" >
    <!-- Inherit from Sensor SimpleProcess where the outputs are defined -->
    <sml:typeOf xlink:title="swebridge:process:powerManagement"/>
    <sml:configuration>
      <sml:Settings>
        <!-- Connect the ADC to the fields -->
        <sml:setValue ref="parameters/powerIndex">12</sml:setValue>
        <sml:setValue ref="parameters/power">OFF</sml:setValue>
```

```
          </sml:Settings>
        </sml:configuration>
      </sml:SimpleProcess>
    </sml:component>
  </sml:ComponentList>
 </sml:components>
 <sml:connections>
  <sml:ConnectionList>
   <!-- Connects the instrument output with the selector component input-->
   <sml:connection>
    <sml:Link>
     <sml:source ref="components/PowerOnSensor/outputs/dataOut"/>
     <sml:destination ref="components/takeSample/inputs/dataIn"/>
    </sml:Link>
   </sml:connection>
   <sml:connection>
    <sml:Link>
     <sml:source ref="components/takeSample/outputs/dataOut"/>
     <sml:destination ref="components/insertResult/inputs/dataIn"/>
    </sml:Link>
   </sml:connection>
   <sml:connection>
    <sml:Link>
     <sml:source ref="components/takeSample/outputs/dataOut"/>
     <sml:destination ref="components/PowerOffSensor/inputs/dataIn"/>
    </sml:Link>
   </sml:connection>
  </sml:ConnectionList>
 </sml:connections>
</sml:AggregateProcess>
```

Te SWE Bridge Test mission 12 provides an example on how to power on a sensor, and configure a data stream (see figure 11). The first process to be executed is the power on sensor (it is flagged as an `initialization process`). Once the sensor is powered on, a configuration command is sent, configuring the sensor in streaming mode. After the initialization sequence (components in the upper part of the workflow), the take sapmle process is receiving data streams, which are processed and stored in a O&M file.

Listing 20: SWE Bridge test 4 mission

```
<sml:AggregateProcess gml:id="test12Mission">
 <sml:components>
  <sml:ComponentList>
   <sml:component name="PowerOnSensor">
          <sml:SimpleProcess gml:id="PowerOnSensor" >
           <gml:identifier codeSpace="uid">swebridge:powerOnSensor</gml:identifier>
           <sml:typeOf xlink:title="swebridge:process:powerManagement"/>
           <sml:configuration>
            <sml:Settings>
             <sml:setValue ref="parameters/powerIndex">12</sml:setValue>
             <sml:setValue ref="parameters/delayAfter">10</sml:setValue>
             <sml:setValue ref="parameters/power">ON</sml:setValue>
             <sml:setValue ref="parameters/executionModes/initialization">true</sml:setValue>
            </sml:Settings>
           </sml:configuration>
          </sml:SimpleProcess>
         </sml:component>
```

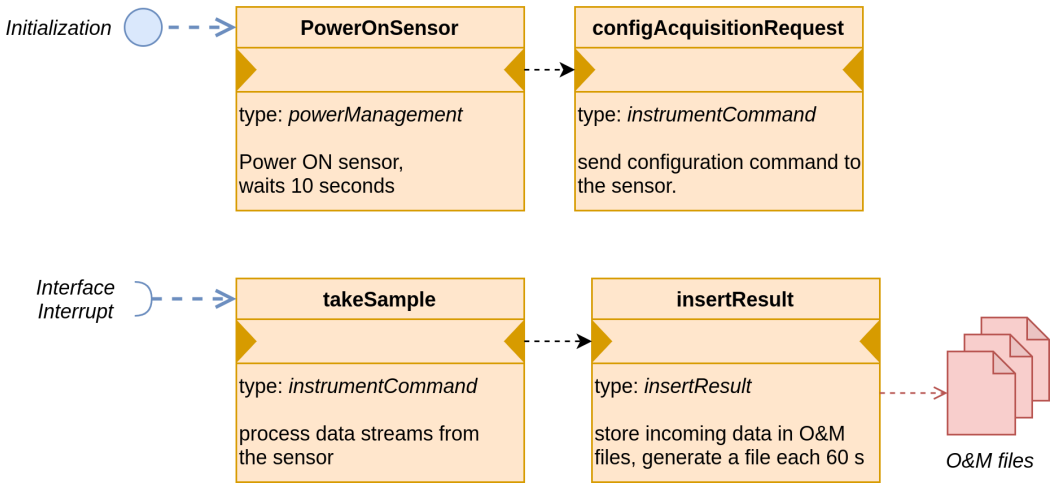


Figure 11: SWE Bridge test mission 12. The sensor is power on and configured during the initialization. Afterwards the data streams from the sensor are stored in O&M files.

```
<sml:component name="configAcquisitionRequest">
  <sml:SimpleProcess gml:id="configDataInstance">
    <gml:identifier codeSpace="uid">test12:configData:1</gml:identifier>
    <sml:typeOf xlink:title="test12:configData"/>
  </sml:SimpleProcess>
</sml:component>
<sml:component name="takeSample">
  <sml:SimpleProcess gml:id="takeSampleInstance" >
    <gml:identifier codeSpace="uid">test12:queryData:1</gml:identifier>
    <sml:typeOf xlink:title="test12:queryData"/>
    <sml:configuration>
      <sml:Settings>
        <sml:setStatus ref="outputs/dataOut/data/response/salinity">disabled</sml:setStatus>
        <sml:setStatus ref="outputs/dataOut/data/response/conductivity">disabled</sml:setStatus>
      </sml:Settings>
    </sml:configuration>
  </sml:SimpleProcess>
</sml:component>
<sml:component name="insertResult">
  <sml:SimpleProcess gml:id="insertResultProcessName">
    <gml:identifier codeSpace="uid">swebridge:modules:insertResult:1</gml:identifier>
    <sml:typeOf xlink:title="swebridge:modules:insertResult"/>
    <sml:configuration>
      <sml:Settings>
        <sml:setValue ref="parameters/msecTimestamp">true</sml:setValue>
        <sml:setValue ref="parameters/recordingTime">60.0</sml:setValue>
        <sml:setValue ref="parameters/template">swe_bridge_test12</sml:setValue>
        <sml:setValue ref="parameters/decimalPrecision">6</sml:setValue>
      </sml:Settings>
    </sml:configuration>
  </sml:SimpleProcess>
</sml:component>
</sml:ComponentList>
</sml:components>
```

```
  <sml:connections>
    <sml:ConnectionList>
      <sml:connection>
        <sml:Link>
          <sml:source ref="components/PowerOnSensor/outputs/dataOut"/>
          <sml:destination ref="components/configAcquisitionRequest/inputs/dataIn"/>
        </sml:Link>
      </sml:connection>
      <sml:connection>
        <sml:Link>
          <sml:source ref="components/takeSample/outputs/dataOut"/>
          <sml:destination ref="components/insertResult/inputs/dataIn"/>
        </sml:Link>
      </sml:connection>
    </sml:ConnectionList>
  </sml:connections>
</sml:AggregateProcess>
```

4 Hydrophone SensorML Description

Due to its intrinsic characteristics, hydrophones are much more complex than regular scientific instruments. In order to interface a hydrophone with the SWE Bridge a Hydrophone SensorML Description (HSD) is required. It is a special type of Sensor Deployment File that aims to abstract a hydrophone. Its UML model is depicted in figure 12. In addition to metadata already described in SDF, an HSD also includes the hydrophone characteristics and additional metadata. Using these characteristics, the SWE Bridge deals with acoustic data streams in real time and provide underwater sound levels in real-time, WAV recordings and more.

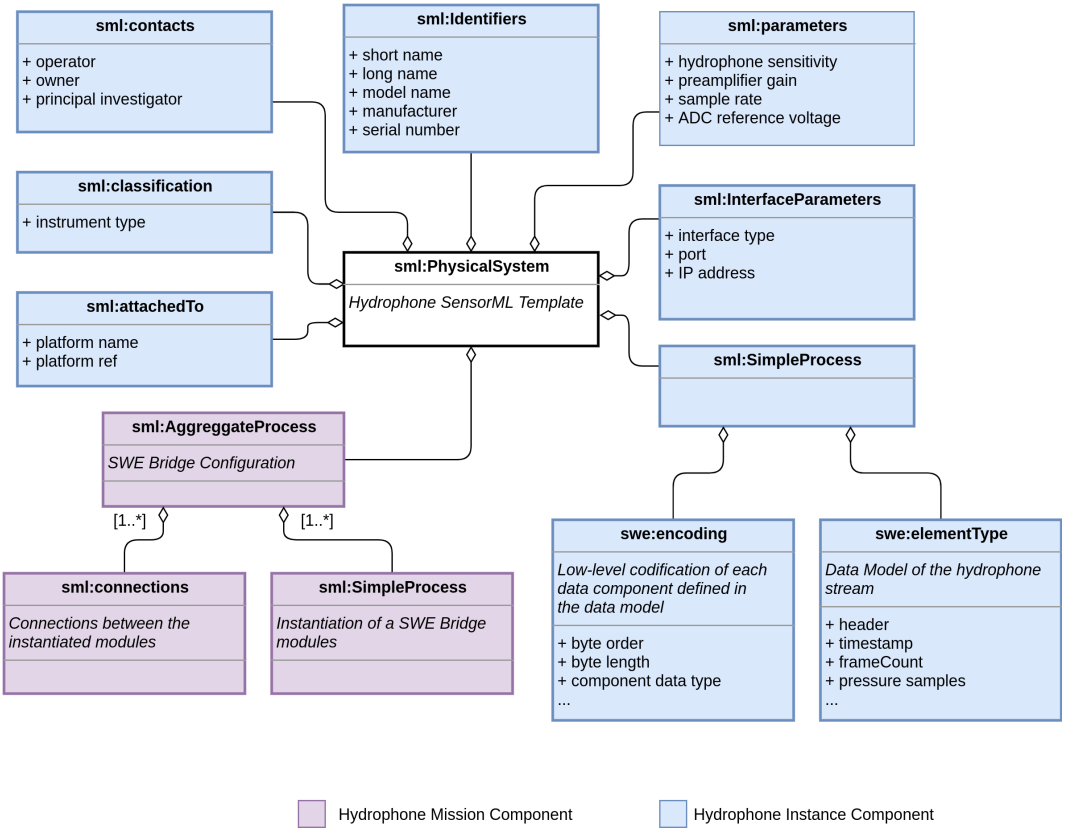


Figure 12: Hydrophone SensorML Description UML diagram

Figure 12 shows the UML data model of a HSD. Note that it includes significantly more metadata than a traditional SDF.

4.1 Hydrophone Characeristics

When processing raw acoustic data, it is very important to provide information about the hydrophone signal acquisition stage. This information includes the

hydrophone sensitivity, the preamplifier gain, the ADC reference voltage, the ADC number of bits and the sample rate. This information has to be encoded using the `sml:capabilities` or `sml:paremeter` element:

Listing 21: Defining the hydrophone's signal acquisition stage within a SensorML file using parameters

```
<sml:parameter name="hydrophone_sensitivity">
  <swe:Quantity
     definition="http://mmisw.org/ont/ioos/passive_acoustic_metadata/hydrophone_sensitivity">
    <swe:label>hydrophone sensitivity</swe:label>
    <swe:uom code="dB"/>
    <swe:value>-192</swe:value>
  </swe:Quantity>
</sml:parameter>
<sml:parameter name="preamplifier_gain">
  <swe:Quantity
    definition="http://mmisw.org/ont/ioos/passive_acoustic_metadata/preamplifier_gain">
    <swe:uom code="dB"/>
    <swe:value>20</swe:value>
  </swe:Quantity>
</sml:parameter>
<sml:parameter name="reference_voltage">
  <swe:Quantity definition="reference_voltage">
    <swe:uom code="V">
    <swe:value>2.5</swe:value>
  </swe:Quantity>
</sml:parameter>
<sml:parameter name="sampling_rate">
  <swe:Quantity
     definition="http://mmisw.org/ont/ioos/passive_acoustic_metadata/sample_rate">
    <swe:uom code="Hz"/>
    <swe:value>10000</swe:value>
  </swe:Quantity>
</sml:parameter>
```

Note that the definition attribute of each parameter is pointing to an online vocabulary. These URLs are used as identifiers for these parameters and should not be changed.

4.1.1 Hydrophone Stream Modelling

Hydrophone streams use high-frequency binary communications. The following example shows how to model in SensorML the stream from a NAXYS Ethernet hydrophone (figure 13).

The Data Stream from a hydrophone depicted at figure 13 can be encoded as a binary sensor command. Note that it is composed of a header byte, a byte count, and an array of pressure samples. Thus, stream has repeating and non-repeating elements. To model this, a `swe:DataRecord` is used, with a nested `swe:DataArray`:

Listing 22: NAXYS Ethernet hydrophone data stream

```
<swe:DataStream>
  <swe:elementType name="dataModel">
```

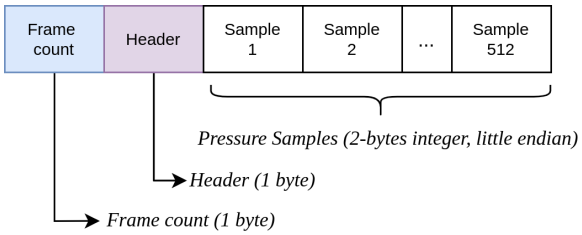


Figure 13: NAXYS Ethernet hydrophone data stream

```
    <swe:DataRecord>
      <swe:field name="frameCount">
        <swe:Count/>
      </swe:field>
      <swe:field name="header">
        <swe:Count/>
      </swe:field>
      <swe:field name="array">
        <swe:DataArray>
          <swe:elementCount xlink:href="#arrayCount"/>
          <swe:elementType name="samples">
            <swe:Count
              definition="http://vocab.nerc.ac.uk/collection/P07/current/CFSN0310/"/>
          </swe:elementType>
        </swe:DataArray>
      </swe:field>
    </swe:DataRecord>
  </swe:elementType>
  <swe:encoding>
    <swe:BinaryEncoding byteOrder="littleEndian" byteEncoding="raw" byteLength="1026">
      <swe:member>
        <swe:Component
          dataType="http://www.opengis.net/def/dataType/OGC/0/unsignedByte"
          ref="dataModel/header"/>
      </swe:member>
      <swe:member>
        <swe:Component
          dataType="http://www.opengis.net/def/dataType/OGC/0/unsignedByte"
          ref="dataModel/frameCount"/>
      </swe:member>
      <swe:member>
        <swe:Component
          dataType="http://www.opengis.net/def/dataType/OGC/0/signedShort"
          ref="dataModel/array/samples"/>
      </swe:member>
    </swe:BinaryEncoding>
  </swe:encoding>
  <swe:values/>
</swe:DataStream>
```

Note that the length of the array is not defined in the `swe:DataStream`, but referenced. It can be specified as a parameter using the following code:

```
<sml:parameter name="p">
    <swe:Count id="arrayCount">
        <swe:value>512</swe:value>
```

```
    </swe:Count>
</sml:parameter>
```

4.2 Hydrophone Mission

When interfacing hydrophones with the SWE Bridge usually real-time underwater noise levels are desired and/or acoustic recordings are desired. In figure 14 both are done at the same time.

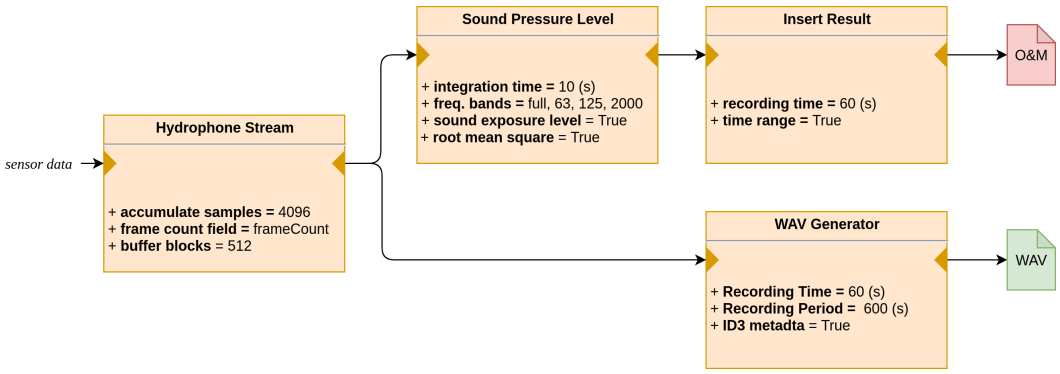


Figure 14: Sensor Deployment File UML diagram

The mission depicted in figure 14 processes data streams incoming from a hydrophone using the hydrophone stream module. This module implements a ring buffer and accumulates samples upon arrival, until a defind number of samples are accumulated (in the example 4096). Then, these pressure samples are sent to the Sound Pressure Level (SPL) module. This module calculates SPL levels according to the specifications of the Marine Strategy Framework Directive (MSFD) in the full dataset and in a set of third-octave band frequencies (in the example the bands centered at 63, 125 and 2000 Hz). The SPL levels are stored in a O&M file using the insert result module. Additionally, acoustic data is also sent to the WAV Generator process, which generates acoustic recordings. This mission can be encoded within a HSD using the following excerpt:

Listing 23: Hydrophone mission with real-time SPL measurements and WAV recordings

```
<sml:AggregateProcess gml:id="naxys_ethernetMission">
  <sml:components>
    <sml:ComponentList>
      <sml:component name="HighFreqStream">
        <sml:SimpleProcess gml:id="HighFreqStream" >
          <!-- Inherit from Sensor SimpleProcess where the outputs are defined -->
          <sml:typeOf xlink:title="naxys_ethernet:dataStream"/>
          <sml:configuration>
            <sml:Settings>
              <sml:setValue ref="parameters/accumulateSamples">4096</sml:setValue>
```

```
            <sml:setValue ref="parameters/frameCountField">frameCount</sml:setValue>
                <sml:setValue ref="parameters/bufferBlocks">512</sml:setValue>
          </sml:Settings>
        </sml:configuration>
      </sml:SimpleProcess>
    </sml:component>
        <sml:component name="SoundPressureLevel">
      <sml:SimpleProcess gml:id="SoundPressureLevel" >
        <sml:typeOf xlink:title="swebridge:modules:soundPressureLevel"/>
        <sml:configuration>
          <sml:Settings>
            <sml:setValue ref="parameters/frequencyBands">full 63 125 2000</sml:setValue>
            <sml:setValue ref="parameters/integrationTime">10</sml:setValue>
          </sml:Settings>
        </sml:configuration>
      </sml:SimpleProcess>
    </sml:component>
    <sml:component name="insertResult">
      <sml:SimpleProcess gml:id="insertResult" >
        <gml:identifier codeSpace="uid">swebridge:insertResult:1</gml:identifier>
        <sml:typeOf xlink:title="swebridge:modules:insertResult"/>
        <sml:configuration>
          <sml:Settings>
            <sml:setValue ref="parameters/numberOfMeasures">6</sml:setValue>
            <sml:setValue ref="parameters/template">test13</sml:setValue>
          </sml:Settings>
        </sml:configuration>
      </sml:SimpleProcess>
    </sml:component>
    <sml:component name="WavGenerator">
      <sml:SimpleProcess gml:id="WavGenerator">
        <sml:typeOf xlink:title="swebridge:modules:wavGenerator"/>
        <sml:configuration>
          <sml:Settings>
            <sml:setValue ref="parameters/recordingTime">10.0</sml:setValue>
            <sml:setValue ref="parameters/recordingPeriod">60.0</sml:setValue>
            <sml:setValue ref="parameters/ID3metadata">TRUE</sml:setValue>
          </sml:Settings>
        </sml:configuration>
      </sml:SimpleProcess>
    </sml:component>
  </sml:ComponentList>
</sml:components>
<sml:connections>
  <sml:ConnectionList>
    <sml:connection>
      <sml:Link>
        <sml:source ref="components/HighFreqStream/outputs/dataOut"/>
        <sml:destination ref="components/SoundPressureLevel/inputs/dataIn"/>
      </sml:Link>
    </sml:connection>
    <sml:connection>
      <sml:Link>
        <sml:source ref="components/HighFreqStream/outputs/dataOut"/>
        <sml:destination ref="components/WavGenerator/inputs/dataIn"/>
      </sml:Link>
    </sml:connection>
    <sml:connection>
      <sml:Link>
```

```
        <sml:source ref="components/SoundPressureLevel/outputs/dataOut"/>
        <sml:destination ref="components/insertResult/inputs/dataIn"/>
      </sml:Link>
    </sml:connection>
  </sml:ConnectionList>
 </sml:connections>
</sml:AggregateProcess>
```

Note that the high frequency stream module does not inherit from a module. Instead, its `sml:typeOf` references a simple process previously defined, which inherits from the hydrophone stream module. This is the same behaviour of the instrument command module.

5 SWE Bridge Modules

In this section a comprehensive list of the modules implemented within the SWE Bridge is provided, with a detailed explanation of their parameters and possible configurations.

5.1 Instrument Command

Module to send and/or receive commands from an instrument. It is required to have at least one input and/or one output. The input encodes the command to be sent to the instrument while output describes the expected response. In the case of streaming instruments only an output without input needs to be defined. In the case of commands without response, only an input needs to be defined.

5.1.1 Referencing a Command

The instrument module is usually not instantiated directly, but through a sensor command (see section 3.4.1): the `sml:typeOf` within the mission points to a sensor command, and the `sml:typeOf` from the command points to the module:

Listing 24: Examlpe on instantiate an instrument command module

```
<!-- Sensor Command in the sensor instance -->
<sml:SimpleProcess gml:id="dataStreamID">
    <!-- Define the command as instrumentCommand module -->
    <sml:typeOf xlink:title="swebridge:modules:instrumentCommand"/>
    <!-- give an identifier to this command -->
    <gml:identifier codeSpace="uid">test01:dataStream</gml:identifier>
    <sml:outputs>
     ...
    </sml:outputs>
</sml:SimpleProcess>
...
<!-- Sensor Mission -->
<sml:AggregateProcess gml:id="sensorMission">
  <sml:components>
    <sml:ComponentList>
        <sml:component name="processStream">
            <sml:SimpleProcess gml:id="processStream" >
                <!-- Inherit from previously declared command -->
                <sml:typeOf xlink:title="test01:dataStream"/>
                <sml:configuration>
                    <sml:Settings>
                    ...
                    </sml:Settings>
                </sml:configuration>
            </sml:SimpleProcess>
        </sml:component>
        ...
        </sml:ConnectionList>
  </sml:connections>
</sml:AggregateProcess>
```

5.1.2 Disabling Fields

It is common that a sensor's data stream has more information than required. Maybe some information is not desired for a particular application, or it is just junk data. Within the instrument command module, any field of a sensor data stream can be disabled using the `sml:setStatus` element within the process' `sml:Settings`. Note that by default all fields within a data stream are enabled. The valid values for `sml:setStatus` element are "enabled" and "disabled". In order to reference a field in the response, its path has to be constructed as follows:

```
outputs/<output's name>/data/<elementType's name>/<field's name>
```

In example, assuming an instrument command process that is referencing the sensor command described in listing 12, the field "depth" can be disabled using the following line:

Listing 25: Examlpe on instantiate an instrument command module

```
<sml:setStatus ref="outputs/dataOut/data/response/depth">disabled</sml:setStatus>
```

The instrument command module has the following parameters:

- Timeout -

Reference: parameters/timeout
Data type: Quantity
Default value: 2 (seconds)
Description: If set, this parameter forces the input buffer size to a certain amount of bytes. If not specified it is automatically configured according to the response

```
<sml:setValue ref="parameters/timeout">10</sml:setValue>
```

- Buffer Size -

Reference: parameters/bufferSize
Data type: Count
Default value: (variable)
Description: This parameter allows the user to manually set the input buffer for the process. If not set, the SWE Bridge automatically calculates the required size.

```
<sml:setValue ref="parameters/bufferSize">1024</sml:setValue>
```

- Watchdog -

Reference: parameters/watchdog
Data type: Quantity
Default value: -
Description: If this parameter is set to a positive number (in seconds), a time-out is configured. If the communication's interface does not receive an incoming transmission before the timer expires the Scheduler is stopped. This option is useful to automatically liberate ports in TCP/IP connection if the connection is unexpectedly broken.

```
<sml:setValue ref="parameters/watchdog">5.7</sml:setValue>
```

- Check Data Integrity -

Reference: parameters/checkDataIntegrity
Data type: Boolean
Default value: TRUE
Description: If set to TRUE, the incoming data will be checked for errors (e.g. decimals in Count values, text in Quantity, etc.). In the event of an error the incoming packet is discarded.

```
<sml:setValue ref="parameters/checkDataIntegrity">FALSE</sml:setValue>
```

- Data Record Block -

Reference: parameters/dataRecordBlock
Data type: Boolean
Default value: TRUE
Description: If this flag is set, the DataRecord Structures will be understood as data blocks, so a block separator is going to be added at the end of each DataRecord containing DataComponents (Quantity, Text, Count, etc.). This setting is useful when several data blocks are sent in a single command. This behaviour is not documented in the SWE Data Common Standard, however it been found in many real-world sensors.

```
<sml:setValue ref="parameters/dataRecordBlock">FALSE</sml:setValue>
```

5.1.3 Burst Mode

The burst mode is designed to perform multiple queries to an instrument in a short period of time and sleep for a longer period afterwards. Its operation is depicted

in figure 15. To activate the burst mode three different parameters need to be set: burst measures, burst period and burst cycle.

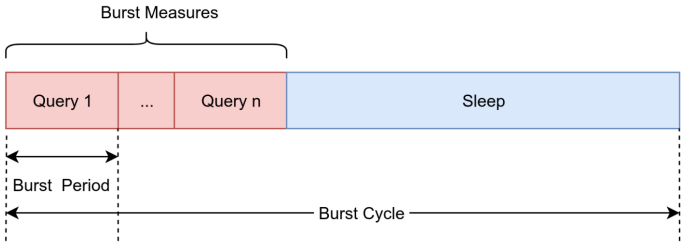


Figure 15: Burst mode sequence

- Burst Measures -

Reference: parameters/burstMeasures
Data type: Count
Default value: -
Description: Number of measures that will be requested each burst cycle

```
<sml:setValue ref="parameters/burstMeasures">3</sml:setValue>
```

- Burst Period -

Reference: parameters/burstPeriod
Data type: Quantity
Default value: -
Description: Delay between measures in burst modes

```
<sml:setValue ref="parameters/burstPeriod">2.0</sml:setValue>
```

- Burst Cycle -

Reference: parameters/burstCycle
Data type: Quantity
Default value: -
Description: Delay between measures in burst modes

```
<sml:setValue ref="parameters/burstCycle">60.0</sml:setValue>
```

5.2 Insert Result

This module is used to store data into O&M files using the *InsertResult* operation.

- Measure Buffer Size -

Reference: parameters/measureBufferSize
Data type: Count
Default value: 1024
Description: This parameter controls the size of the process internal buffer. It is only required to modify when using instruments with very large responses (e.g. ADCPs)

```
<sml:setValue ref="parameters/measureBufferSize">4096</sml:setValue>
```

- Recording Time -

Reference: parameters/recordingTime
Data type: Quantity
Default value: -
Description: This parameter controls the periodicity (in seconds) of the *InsertResult* file generation (fiels are only generated if there are acquired measures). If set to a negative value files are not created based on time.

```
<sml:setValue ref="parameters/recordingTime">300.0</sml:setValue>
```

- Max File Size -

Reference: parameters/maxFileSize
Data type: Count
Default value: 0
Description: If set to a value greater than 0 the output file generation is controlled by size instead of time

```
<sml:setValue ref="parameters/maxFileSize">4096</sml:setValue>
```

- EXI Encoding -

Reference: parameters/EXIencoding
Data type: Boolean
Default value: False

Description: If set to true *InsertResult* files will be encoded in EXI format instead of XML (byte packed without schema).

```
<sml:setValue ref="parameters/EXIencoding">TRUE</sml:setValue>
```

- Temporal Path -

Reference: parameters/temporalPath
Data type: Text
Default value: temp
Description: Path where the module will store temporal files. This files contain data acquired but not yet arranged in the *InsertResult* format.

```
<sml:setValue ref="parameters/temporalPath">mypath/example</sml:setValue>
```

- Output Path -

Reference: parameters/outputPath
Data type: Text
Default value: temp
Description: Path where the *InsertResult* files will be stored.

```
<sml:setValue ref="parameters/outputPath">mypath/example</sml:setValue>
```

- Template -

Reference: parameters/template
Data type: Text
Default value: -
Description: Template identifier for the SOS server. This template shall match with the template identifier from the *InsertResultTemplate* operation where the data model is defined

```
<sml:setValue ref="parameters/template">template-example-01234</sml:setValue>
```

- Georeference -

Reference: parameters/georeference
Data type: Boolean
Default value: False
Description: If set to true the platform's latitude and longitude will be added

alonside the measured parameters.

```
<sml:setValue ref="parameters/georeference">TRUE</sml:setValue>
```

- Altitude -

Reference: parameters/altitude
Data type: Boolean
Default value: False
Description: If set to true the platform's altitude will be added alongside the measured parameters. In underwater applications a negative value will be provided for depth.

```
<sml:setValue ref="parameters/altitude">TRUE</sml:setValue>
```

- Decimal Precision -

Reference: parameters/decimalPrecision
Data type: Count
Default value: 6
Description: Defines the decimal precision of the measurements. The precision will only be applied if some operation on the data is required (e.g. applying calibration, converting from binary to ASCII, etc.).

```
<sml:setValue ref="parameters/decimalPrecision">10</sml:setValue>
```

- Timestamp in Milliseconds -

Reference: parameters/msecTimestamp
Data type: Boolean
Default value: False
Description: If set to true, the timestamp will be stored in milliseconds

```
<sml:setValue ref="parameters/msecTimestamp">TRUE</sml:setValue>
```

- Time Range -

Reference: parameters/timeRange
Data type: Boolean
Default value: False
Description: If set to true, instead of a single date/time value, the timestamp

will be ¡startTime¿/¡endTime¿. Useful for measurements of quantities for more than 1 second

```
<sml:setValue ref="parameters/timeRange">TRUE</sml:setValue>
```

5.3 CSV Generator

This module is used to store data into comma-separated-value (CSV) files.
Parameters:

- Periodicity -

Reference: parameters/periodicity
Data type: Category
Default value: -
Description: Control the periodicity for generating output files. Valid values are ”year”, ”month”, ”day”, ”hour” and ”minute”

```
<sml:setValue ref="parameters/periodicity">hour</sml:setValue>
```

- Delimiter -

Reference: parameters/delimiter
Data type: Text
Default value: ,
Description: Delimiter between measures

```
<sml:setValue ref="parameters/delimiter">,</sml:setValue>
```

.

- End Line -

Reference: parameters/endline
Data type: Text
Default value: <NL >
Description: Delimiter to indicate end of line.

```
<sml:setValue ref="parameters/endline">&#x0A;</sml:setValue>
```

- Georeference -

Reference: parameters/georeference
Data type: Boolean
Default value: False
Description: If set to true the platform's latitude and longitude will be added alongside the measured parameters.

```
<sml:setValue ref="parameters/georeference">TRUE</sml:setValue>
```

- Altitude -

Reference: parameters/altitude
Data type: Boolean
Default value: False
Description: If set to true the platform's altitude will be added alongside the measured parameters. In underwater applications a negative value will be provided for depth.

```
<sml:setValue ref="parameters/altitude">TRUE</sml:setValue>
```

- Decimal Precision -

Reference: parameters/decimalPrecision
Data type: Count
Default value: 6
Description: Defines the decimal precision of the measurements.

```
<sml:setValue ref="parameters/decimalPrecision">10</sml:setValue>
```

- File Prefix -

Reference: parameters/prefix
Data type: Text
Default value: False
Description: Prefix to be added at the output filenames

```
<sml:setValue ref="parameters/prefix">TRUE</sml:setValue>
```

- Timestamp in Milliseconds -

Reference: parameters/msecTimestamp
Data type: Boolean
Default value: False
Description: If set to ture, the timestamp will be stored in milliseconds

```
<sml:setValue ref="parameters/msecTimestamp">TRUE</sml:setValue>
```

- Time Range -

Reference: parameters/timeRange
Data type: Boolean
Default value: False
Description: If set to true, instead of a single date/time value, the timestamp will be <startTime>/<endTime>. Useful for measurements of quantities for more than 1 second

```
<sml:setValue ref="parameters/timeRange">TRUE</sml:setValue>
```

5.4 Linear Calibration

This function applies a linear calibration operation to input measurements. It is useful to convert raw data into a meaningful measurement. The equation linear calibration equation that will be applied is:
$y = m{\cdot}x + a$
Where y is output data, x is input data, m is the slope and a is the offset.

- Add Calibration -

Reference: parameters/addCalibration
Data type: Text
Default value: -
Description: This adds a calibration curve. The value is composed of 3 fields separated by a blank space. The first is the field name where the calibration will be applied, the second the slope and the third parameter is the offset.

```
<sml:setValue ref="parameters/addCalibration">temperatureAnalogOutput 1.01234 -3.2123</sml:setValue>
```

5.5 Subsampling

This module allows to subsample acquired data. It can be useful in communication's constrained platforms where a subsampled data set is transmitted in real-time while the full data set is stored on-board and processed after the platform recovery. Subsampling can be regarded as a filter, which receives all the data, but only allows to pass data in certain conditions. The subsampling strategy can be configured by period (one measure each n seconds) or by ratio (store one of each n measures).

- Sampling Period -

Reference: parameters/samplingPeriod
Data type: Quantity
Default value: -
Description: If set to a value greater than 0, only one measure will be sent from input to output once per period

```
<sml:setValue ref="parameters/samplingPeriod">60.0</sml:setValue>
```

- subsamplingRatio -

Reference: parameters/subsamplingRatio
Data type: Count
Default value: -
Description: If set to a value greater than 0, only one measure will be sent be sent from input to output each n measures received

```
<sml:setValue ref="parameters/subsamplingRatio">10</sml:setValue>
```

5.6 Power Management

This module allows to power on and off instruments. It is only available if the power wrappers have been implemented for the current platform.

- Power Index -

Reference: parameters/powerIndex
Data type: Count
Default value: -
Description: This parameter is the platform's port index where the power operation will be applied

```
<sml:setValue ref="parameters/powerIndex">4</sml:setValue>
```

- Power -

Reference: parameters/power
Data type: Boolean
Default value: -
Description: If set to ON the instrument will be powered up, otherwise it will be powered down. ON is equivalent to TRUE while OFF is equivalent to FALSE.

```
<sml:setValue ref="parameters/power">ON</sml:setValue>
```

- Delay After -

Reference: parameters/delayAfter
Data type: Quantity
Default value: -
Description: If set to a positive value, the system will wait for n seconds before the next process will be executed. This parameter is useful to wait for the instrument's start-up sequence to be completed before sending commands.

```
<sml:setValue ref="parameters/delayAfter">3.0</sml:setValue>
```

Example process using the Power Management module:

```
<sml:SimpleProcess gml:id="PowerOnSensor" >
  <gml:identifier codeSpace="uid">swe_bridge:powerOnSensor</gml:identifier>
  <!-- Inherit from Sensor SimpleProcess where the outputs are defined -->
  <sml:typeOf xlink:title="swe_bridge:process:powerManagement"/>
  <sml:configuration>
    <sml:Settings>
    <!-- Connect the ADC to the fields -->
      <sml:setValue ref="parameters/delayAfter">3.0</sml:setValue>
      <sml:setValue ref="parameters/powerIndex">12</sml:setValue>
      <sml:setValue ref="parameters/power">ON</sml:setValue>
      <sml:setValue ref="parameters/samplingRate">10.00</sml:setValue>
    </sml:Settings>
  </sml:configuration>
</sml:SimpleProcess>
```

5.7 Internal Sensors

This module gathers data from internal sensors. The internal sensor's data should be provided by the appropriate abstraction wrappers in an understandable fashion (e.g. using a string). In the sensor instance a `sml:SimpleProcess` with the data

Requires previous data	no
Returns data	yes
identifier	swebridge:modules:analaogMeasurement

model provided by the internal sensors should be provided. Instrument command module's rules for data modelling and indirect referencing apply to this module.

- Sensor Index -

Reference: parameters/sensorIndex
Data type: Count
Default value: -
Description: Index of the internal sensors (passed to the internal sensors abstraction wrappers).

```
<sml:setValue ref="parameters/sensorIndex">2</sml:setValue>
```

5.8 Analog Measurement

This module reads from an ADC and returns its value as an analog voltage. This module is only operational if the adc wrappers are implemented for the current platform.

- ADCx -

Reference: parameters/ADCx
Data type: Text
Default value: -
Description: The ADCx parameter configures a specific ADC channel (0-7 by default). The value of this parameter is the field where the analog measurmenet will be stored. In the example

```
<sml:setValue ref="parameters/ADC0">temperature</sml:setValue>
```

```
<!-- Defining Sensor Output -->
<sml:SimpleProcess>
  <gml:identifier codeSpace="uid">ExampleADC:analogMeasurement</gml:identifier>
  <!-- Indicate that it is an analog measurement by inheriting analogMeasurement -->
  <sml:typeOf xlink:title="swe_bridge:modules:analogMeasurement"/>
  <sml:outputs>
    <sml:OutputList>
      <!-- Define an output for each of the analog measurements -->
      <sml:output name="temperatureAnalogOutput">
        <swe:Quantity definition="www.linktovocab.com/sea_water_temperature">
          <swe:uom code="mV"/>
```

```
        </swe:Quantity>
      </sml:output>
      <sml:output name="conductivityAnalogOutput">
        <swe:Quantity definition="www.linktovocab.com/sea_water_conductivity">
          <swe:uom code="mV"/>
        </swe:Quantity>
      </sml:output>
      <sml:output name="depthAnalogOutput">
        <swe:Quantity definition="www.linktovocab.com/depth">
          <swe:uom code="mV"/>
        </swe:Quantity>
     </sml:output>
   </sml:OutputList>
 </sml:outputs>
</sml:SimpleProcess>

...
<!-- Defining ADC Module Config -->
<sml:component name="ADCmodule">
  <sml:SimpleProcess gml:id="ADCmodule">
    <gml:identifier codeSpace="uid">takeSample</gml:identifier>
    <!-- Inherit from Sensor SimpleProcess where the outputs are defined -->
    <sml:typeOf xlink:title="ExampleADC:analogMeasurement"/>
    <sml:configuration>
      <sml:Settings>
      <!-- Connect the ADC to the fields -->
        <sml:setValue ref="parameters/samplingRate">5.0</sml:setValue>
        <sml:setValue ref="parameters/ADC0">temperatureAnalogOutput</sml:setValue>
        <sml:setValue ref="parameters/ADC1">conductivityAnalogOutput</sml:setValue>
        <sml:setValue ref="parameters/ADC4">depthAnalogOutput</sml:setValue>
      </sml:Settings>
    </sml:configuration>
   </sml:SimpleProcess>
</sml:component>
```

5.9 Zabbix Sender

This module sends incoming data to a Zabbix monitoring service as a trapper. It requires the external `zabbix_sender.py` python3 script (found in the SWE Bridge tools folder). It only works in UNIX environments.

Note that the variable name as defined in the `sml:field`'s name attribute will be used, so in Zabbix should be registered accordingly.

Parameters:

- Zabbix Sender -

Reference: parameters/zabbixSender
Data type: Text
Default value: -
Description: Path to the `zabbix_sender.py` python script.

```
<sml:setValue ref="parameters/zabbixSender">/opt/zabbix/zabbix_sender.py</sml:setValue>
```

- IP -

Reference: parameters/IP
Data type: Text
Default value: -
Description: IP address or hostname of the zabbix service

```
<sml:setValue ref="parameters/IP">192.168.1.145</sml:setValue>
```

- Port -

Reference: parameters/port
Data type: Count
Default value: -
Description: Port number of the zabbix service

```
<sml:setValue ref="parameters/port">1051</sml:setValue>
```

- Host Name -

Reference: parameters/hostName
Data type: Text
Default value: -
Description: Name of the instrument (as registered in zabbix)

```
<sml:setValue ref="parameters/hostName">MySensorNameAtZabbix</sml:setValue>
```

5.10 Hydrophone Stream

This module processes data from a hydrophone sending acoustic data. This module can only be used in Hydrophone SensorML Descriptions. It implements a ring buffer to accumulate incoming samples until a certain amount of samples is accumulated, then these samples are passed to the next process. Instrument command module's rules for data modelling and indirect referencing apply to this module.

- Accumulate Samples -

Reference: parameters/accumulateSamples
Data type: Count

Default value: -
Description: Number of samples to accumulate before executing next processes. A small amount of samples (several packets) is usually desired to reduce the buffer memory.

```
<sml:setValue ref="parameters/accumulateSamples">4096</sml:setValue>
```

- Buffer Blocks -

Reference: parameters/bufferBlocks
Data type: Count
Default value: -
Description: Number of blocks to be allocated by the circular buffer. Each block has the size of a full data packet. Greater the buffer size the safer it is, but more memory will be used

```
<sml:setValue ref="parameters/bufferBlocks">256</sml:setValue>
```

- Frame Count Field -

Reference: parameters/frameCountField
Data type: Text
Default value: -
Description: This parameter defines which field from the data model is the frame count. The frame count allow to detect missing packets and deal with timing errors.

```
<sml:setValue ref="parameters/frameCountField">frameCount</sml:setValue>
```

5.11 Sound Pressure Level

This module processes acoustic data and provides Sound Pressure Level (SPL) measurements in real time, following the specifications of the Marine Strategy Framework Directive (MSFD) indicator 11 and following best practices on underwater noise [7]. The SPL measurements can be calculated in through the full hydrophone bandwidth, and/or in third-octave frequency bands. Additionally, it is also possible to measure the Sound Exposure Level (SEL) and the Root Mean Square (RMS) pressure.

5.11.1 Theroetical Background

This process estimates the SPL based on the Power Spectral Density (PSD) of the incoming data by means of the Welch or Bartlett's methods. If window option is selected, the Welch method with a Hann window and 50% overlap will be used. Otherwise, the windowless Bartlett's method will be used. If the window is selected, the coherent gain (CG) and the normalized equivalent noise bandwidth (NENBW) corrections will be applied.

The periodogram of a windowed signal (also known as modified periodogram) can be calculated using (1) [8].

$$P'_{xx}(f) = \frac{1}{N \cdot nenbw} \left| \frac{P_w(f)}{CG} \right|^2 = \frac{|P_w(f)|^2}{\sum_N w(n)^2} \tag{1}$$

Being N the number of points in the pressure signal, $P_w(f)$ the DFT transform of the windowed pressure signal $p_w(n)$, and $w(n)$ the window function. Note that *nenbw* and *CG* corrections assume that the input signal is stationary and has a flat spectral response, i.e. white Gaussian noise. In real-world acoustic signals this is rarely the case, so a small error due to windowing side-effects is expected.

One of the weak points of the periodogram as a PSD estimator is its high variance. In order to reduce the variance of each estimate, several periodograms can be averaged using the Bartlett's or Welch's methods. However, when averaging periodograms there is a trade-off between the frequency resolution Δf and the variance reduction [9]. On the other hand, SPLs are usually calculated over large periods of time (usually tens of seconds), so averaging can help to reduce the computational costs in terms of memory and number of operations [10].

The Bartlett's method slices the signal with M samples into K non-overlapping, sequential segments ($M = K \cdot N$). Then it calculates the periodogram for each segment and averages the results (Fig. 16, top) [9]. This approach does not require *CG* nor *nenbw* corrections, but the effect of the spectral leakage is greater.

The Welch method is similar, but the slices are windowed and overlapped (Fig. 16, bottom) [9], [10]. This method has significantly less spectral leakage, but uses the *CG* and *nenbw* corrections, which may also induce errors due to the non-stationary and non-gaussian nature of real-world acoustic signals. Additionally, an increase of computational cost due to the FFT overlapping is expected.

Then, the power spectral density can be estimated using the Welch's method or the Bartlett's method by averaging K periodograms:

$$\overline{P}_{xx}(f) = \frac{1}{K} \sum_{i=0}^{i=K-1} P'_{xx_i}(f) \tag{2}$$

Where $\overline{P}_{xx}(f)$ is the power spectral density estimation, K is the number of periodograms being averaged and $P_{xx_i}(f)$ is the periodogram of the ith data seg-

ment. The number of segments to be averaged depends on the length of each data segment N and on the overlapping [10].

Once the power spectral density has been estimated, the SPL value within a frequency band over a time interval $T' = M/f_s$ can be calculated with (3).

$$SPL_{f,N} = 10\log_{10}\left(\frac{2}{N}\sum_{f_l}^{f_h}\overline{P}_{xx}(f)\right)[dB\ re\ \mu Pa] \tag{3}$$

Being f_l and f_h the low / high limit frequencies of the desired third-octave band and N is the number of samples within each FFT segment. Since the PSD of a real signal is symmetric, the negative frequencies are redundant and can be omitted [11]. However, to consider their energy contribution, a factor of 2 has to be applied to the sum of the positive frequency bins. Note that the result of (3) depends on multiple parameters, including the total number of samples M, the number of samples in each data segment N, the overlapping between data segments and the window function.

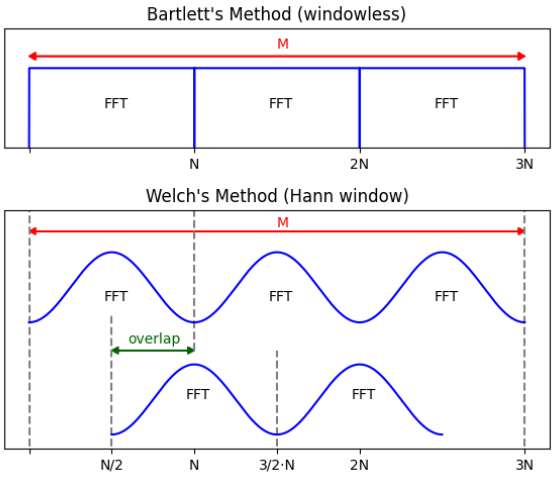


Figure 16: Welch's and Bartlett's methods for power spectral density estimation. The Bartlett's method uses sequential, non-overlapped, windowless data segments. The Welch method uses overlapped windowed data segments, in this example it uses a Hann window and a 50% overlap.

In (3) several adjacent bins are added to approximate a third-octave band. However, there is only a finite number of frequency bins and f_l and f_h may not coincide with them. For instance, the pressure signal's power spectrum calculated over a time period of 1 second has a bin width of 1 Hz. If the SPL in the third-octave band centered at 63 Hz is calculated, the edge frequencies are 56.6 Hz and

70.8 Hz, which are not aligned with the frequency bins. In order to correct this misalignment a bandwidth correction needs to be applied, as shown in (4) [12].

$$SPL'_{f,N} = SPL_{f,N} - 10\log_{10}\frac{BW_{real}}{BW_{ideal}} \tag{4}$$

Where $SPL'_{f,N}$ is the sound pressure level value with the bandwidth correction, BW_{ideal} is the ideal bandwidth of the third octave band (f_h - f_l) and BW_{real} is the real bandwidth summed in (3).

Although SPL is the main parameter when measuring ocean sound, some other parameters may also be of interest, such as the Sound Exposure Level (SEL) and the Root Mean Squared (RMS) pressure. Since both parameters use similar calculations as the SPL, they are also implemented within the SWE Bridge to provide extra information at very little computational cost.

SEL is a measure of the integral of the square of the sound pressure over a stated time interval or event expressed in decibels [13]. This parameter is closely linked with the SPL value, but making the time interval explicit. It can be easily derived from an SPL value using (5) [12].

$$SEL_T = SPL'_{f,N} + 10\log_{10}\left(\frac{T'}{T_0}\right)[dB\ 1\ \mu Pa^2 s] \tag{5}$$

Being T' the time window over which the SPL value was calculated (proportional to M) and $T_0 = 1$ second is the time reference.

The P_N is the RMS value of the pressure signal over a segment of length N, calculated using (6):

$$P_N = \sqrt{\frac{1}{N}\sum_N p(n)^2}[Pa] \tag{6}$$

In order to maintain the same time window as SPL and SEL values, several RMS values are averaged, as stated in (7).

$$\overline{P}_{N,M} = \frac{1}{M}\sum_M P_N[Pa] \tag{7}$$

Being $\overline{P}_{N,M}$ the mean of M RMS values. The SWE Bridge provides the averaged mean of several RMS values for two reasons: to maintain the same time window used in SPL and SEL calculations, and to prevent the use of very large buffer which may result in memory overflow and excessive computational cost.

5.11.2 Module Parameters

- Frequency Bands -

Reference: parameters/frequencyBands
Data type: Text
Default value: -
Description: List of third-octave bands (center frequency) to calculate SPL values using equation 4. If "full" is used, the whole bandwidth will also calculated. The bands should be separated by a space

```
<sml:setValue ref="parameters/frequencyBands">full 63 125 2000</sml:setValue>
```

- Sound Exposure Level -

Reference: parameters/soundExposureLevel
Data type: Boolean
Default value: False
Description: If set to true, the SEL values wil also be calculated using equation 5

```
<sml:setValue ref="parameters/soundExposureLevel">True</sml:setValue>
```

- Root Mean Square -

Reference: parameters/soundExposureLevel
Data type: Boolean
Default value: False
Description: If set to true, the SEL values wil also be calculated using equation 7

```
<sml:setValue ref="parameters/soundExposureLevel">True</sml:setValue>
```

- Window -

Reference: parameters/window
Data type: Boolean
Default value: False
Description: Whether to use a window function (Welch's method) or not (Bartlett's method). If set to true a Hann window with 50% overlap will be used.

```
<sml:setValue ref="parameters/window">True</sml:setValue>
```

- FFT Time -

Reference: parameters/fftTime
Data type: Quantity
Default value: 1.0
Description: Length (in seconds) of each FFT segment. This parameter is of the outmost importance, since it will define the frequency resolution (Δf). In the equation it is proportional to N

```
<sml:setValue ref="parameters/fftTime">2.0</sml:setValue>
```

- Integration Time -

Reference: parameters/integrationTime
Data type: Quantity
Default value: 10.0
Description: Length (in seconds) over which the SPL will be calculated. It must be a multiple of FFT Time. It determines how many periodograms will be averaged. It is proportional to the total number of samples M .

```
<sml:setValue ref="parameters/integrationTime">20.0</sml:setValue>
```

5.12 WAV Generator

This module stores incoming acoustic data into WAV files. This module can only be used in Hydrophone SensorML Descriptions. WAV files can be generated continuously or only a subset of data may be generated (e.g. one minute out of ten). This module also as the option of embedding hydrophone metadata as user-defined tags within the generated files using the ID3 informal standard [14].

- Recording Time -

Reference: parameters/recordingTime
Data type: Quantity
Default value: -
Description: This parameter controls the length (in seconds) of the generated WAV files.

```
<sml:setValue ref="parameters/recordingTime">60.0</sml:setValue>
```

- Temporal Folder -

Reference: parameters/outputFolder
Data type: Text
Default value: temp
Description: Path where the temporal WAV files will be generated.

```
<sml:setValue ref="parameters/outputFolder">myCustomWavFolder</sml:setValue>
```

- Output Folder -

Reference: parameters/outputFolder
Data type: Text
Default value: wavs
Description: Path where the WAV files will be stored.

```
<sml:setValue ref="parameters/outputFolder">myCustomWavFolder</sml:setValue>
```

- Prefix -

Reference: parameters/prefix
Data type: Text
Default value: -
Description: Prefix that will be used at the beginning of each filename.

```
<sml:setValue ref="parameters/prefix">myHydrophoneName</sml:setValue>
```

- ID3 Metadata -

Reference: parameters/ID3metadata
Data type: Boolean
Default value: True
Description: If active, hydrophone information will be stored within the WAV file as ID3 user-defined tags.

```
<sml:setValue ref="parameters/ID3metadata">False</sml:setValue>
```

- Recording Period -

Reference: parameters/recordingPeriod
Data type: Quantity

Code	Meaning	Example
YYYY	Year (4 digits)	2020
YY	Year (2 digits)	20
MM	Month (2 digits)	12
DD	Day (2 digits)	31
hh	Hour (2 digit)	12
mm	Minute (2 digit)	31
ss	Seconds (2 digit)	59

Table 3: Timestamp pattern symbols

Default value: -
Description: If set to a number greater than recording time, only one file (with length recording time) will be generated every recording period seconds. Useful to configure a recording cycle to save space in the file system.

```
<sml:setValue ref="parameters/recordingPeriod">600</sml:setValue>
```

5.13 WAV Reader

This module reads acoustics recordings in WAV format. It is useful for analyzing acoustic recordings after they have been acquired. Since WAV files do not contain timestamp information, it is important that all the files have a machine-readable timestamp in their filenames. The timestamp pattern can be defined following using the codes in table 3.

In example, a compact timestamp:

```
WAV File:  MyHydrophone_20201231_123159.wav
Pattern :  MyHydrophone_YYYYMMDD_hhmmss.wav
```

Or an expanded timestamp:

```
WAV File:  MyHydrophone_2020-12-31_12:31:59.wav
Pattern :  MyHydrophone_YYYY-MM-DD_hh:mm:ss.wav
```

- Accumulate Sample -

Reference: parameters/accumulateSamples
Data type: count
Default value: 21000
Description: Number of samples to be read before sending them to further processes.

```
<sml:setValue ref="parameters/accumulateSamples">4096</sml:setValue>
```

- Input Files -

Reference: parameters/inputFiles
Data type: Text
Default value: -
Description: Pattern the WAV files to be analyzed

```
<sml:setValue ref="parameters/inputFiles">pathToWavFiles</sml:setValue>
```

- Timestamp Pattern -

Reference: parameters/Timestamp Pattern
Data type: Text
Default value: -
Description: Path containing the WAV files to be analyzed

```
<sml:setValue ref="parameters/Timestamp Pattern"></sml:setValue>
```

- Timezone -

Reference: parameters/timezone
Data type: Quantity
Default value: 0.0
Description: Timezone as a floating point value. In example 2.0 is CEST (UTC+2), -5.0 is Eastern Standard Time (UTC−5), etc.

```
<sml:setValue ref="parameters/timezone">2.0</sml:setValue>
```

References

[1] T. O'Reilly, "OGC® PUCK Protocol Standard Version 1.4," tech. rep., Open Geospatial Consortium, Wayland, MA, 01778, USA, 2012.

[2] M. Botts and A. Robin, "OGC SensorML: Model and XML Encoding Standard 2.0," tech. rep., Open Geospatial Consortium, Wayland, MA, USA, 2014.

[3] A. Robin, "OGC SWE Common Data Model Encoding Standard," tech. rep., Open Geospatial Consortium, Wayland, MA, USA, 2011.

[4] A. Bröring, C. Stasch, and J. Echterhoff, "OGC Sensor Observation Service," tech. rep., Open Geospatial Consortium, Wayland, MA, USA, 2012.

[5] S. Cox, "Observations and Measuremets-XML Implementation," tech. rep., Open Geospatial Consortium, Wayland, MA, USA, 2011.

[6] J. Schneider, T. Kamiya, D. Peintner, and R. Kyusakov, "Efficient XML Interchange (EXI) Format 1.0," tech. rep.

[7] R. Dekeling, M. Tasker, A. Van der Graaf, M. Ainslie, M. Andersson, M. André, J. Borsani, K. Brensing, M. Castellote, D. Cronin, J. Dalen, T. Folegot, R. Leaper, J. Pajala, P. Redman, S. Robinson, P. Sigray, G. Sutton, F. Thomsen, S. Werner, D. Wittekind, and J. Young, "Monitoring Guidance for Underwater Noise in European Seas, Part II: Monitoring Guidance Specifications," tech. rep., Publications Office of the European Union, Luxembourg, 2014.

[8] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing*. Upper Saddle River, NJ, USA: Pearson Prentice Hall, 4 ed., 2007.

[9] P. Stoica and R. Moses, *Spectral analysis of signals*. Upper Saddle River, NJ, USA: Pearson Prentice Hall, 2005.

[10] P. Welch, "The use of fast Fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms," *IEEE Transactions on Audio and Electroacoustics*, vol. 15, no. 2, pp. 70–73, 1967.

[11] G. Heinzel, A. Rüdiger, and R. Schilling, "Spectrum and Spectral Density Estimation by the Discrete Fourier Transform(DFT), Including a Comprehensive List of Window Functions and Some New At-Top Windows," tech. rep., Max-Planck-Institut fur Gravitationsphysik, Hannover, Germany, 2002.

[12] U. K. Verfuß, M. Andersson, T. Folegot, J. Laanearu, R. Matuschek, J. Pajala, P. Sigray, J. Tegowski, and J. Tougaard, "Bias Standards for noise measurements. Background information, Guidelines and Quality Assurance," tech. rep., 2015.

[13] S. P. Robinson, P. A. Lepper, and R. A. Hazelwood, "Good Practice Guide for Underwater Noise Measurement," Tech. Rep. 133, National Measurement Office, Marine Scotland, The Crown Estate, Teddington, England, 2014.

[14] M. Nilsson, "ID3," 2000.

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament d'Enginyeria Electrònica